

## **General Disclaimer**

### **One or more of the Following Statements may affect this Document**

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.
- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.
- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.
- This document is paginated as submitted by the original source.
- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

Phase II  
Final  
Report

NASA CR-

144528

October 1975

Volume I

Design Considerations  
for a Scheduling and  
Resource Allocation  
Computer System

**Scheduling  
Language and  
Algorithm  
Development  
Study**

(NASA-CR-144528) SCHEDULING LANGUAGE AND  
ALGORITHM DEVELOPMENT STUDY. VOLUME 1,  
PHASE 2: DESIGN CONSIDERATIONS FOR A  
SCHEDULING AND RESOURCE ALLOCATION SYSTEM  
Final Report (Martin Marietta Corp.) 122 p G3/61

N76-11751

Unclas  
01859

**MARTIN MARIETTA**

## FOREWORD

---

This is the final report for Phase II of the Scheduling Language and Algorithm Development Study (NAS9-13616). It is contained in three volumes. The objectives of Phase II were to implement prototypes of the Scheduling Language called PLANS and the scheduling module library that were designed and specified in Phase I.

Volume I of this report contains data and analyses related to a variety of algorithms for solving typical large-scale scheduling and resource allocation problems. The capabilities and deficiencies of various alternative problem solving strategies are discussed from the viewpoint of computer system design.

Volume II is an introduction to the use of the Programming Language for Allocation and Network Scheduling (PLANS). It is intended as a reference for the PLANS programmer.

Volume III contains the detailed specifications of the scheduling module library as implemented in Phase II. This volume extends the Detailed Design Specifications previously published in the Phase II Interim report (April 1975).

## CONTENTS

---

	<u>Page</u>
1.0 Introduction . . . . .	1
2.0 Fully Computerizable Problem-Solving Strategies . . .	5
2.1 Mathematical Programming Techniques . . . . .	5
2.1.1 Direct Application . . . . .	6
2.1.2 Application via Mathematical Decomposition . . . . .	15
2.2 Heuristic Algorithms . . . . .	17
2.2.1 Problems with Dominating Network Constraints . . . . .	17
2.2.2 Problems without Dominating Network Constraints . . .	21
2.2.3 Integrated Heuristic Programs for Large Problems . . .	25
2.3 The Problem Model - Algorithm Gap . . . . .	31
3.0 Human Participation in Complex Scheduling Tasks . . .	33
3.1 Methods for Allocating Roles in Man-Computer Systems .	33
3.2 Paper Simulation of Activity Scheduling Problem . . .	35
3.2.1 Introduction . . . . .	35
3.2.2 The Activity Scheduling Problem Statement . . . . .	35
3.2.3 Results . . . . .	42
3.3 Paper Simulation of Project Scheduling Problem . . . .	43
3.3.1 Introduction . . . . .	43
3.3.2 The Project Scheduling Problem Statement . . . . .	43
3.3.3 General Description of the Hypothetical Scheduling System . . . . .	45
3.3.4 Results . . . . .	51
3.4 A Man-Computer Strategy for Solving a Large Class of Scheduling Problems . . . . .	60
3.4.1 Concept . . . . .	60
3.4.2 Functional Design . . . . .	61
3.4.3 Implementation . . . . .	67
3.4.4 Computational Experience . . . . .	87
3.5 A System Configuration Compatible with the Man-Computer Strategy . . . . .	104
3.5.1 Dialog Methods for Scheduling . . . . .	104
3.5.2 Host/Mini-Computer Configuration for Interactive Scheduling . . . . .	111

## FIGURES

---

	<u>Page</u>
2-1 Problem Characteristics Amenable to Mathematical Programming . . . . .	7
2-2 Tire Facility Job Network . . . . .	10
2-3 Portion of Task Flow Diagram of Spacelab Operations . .	18
2-4 Top Level Logic Diagram of Time-Progressive Heuristics .	20
2-5 Top Level Logic Diagram of Time-Transcendent Heuristics . . . . .	23
2-6 Resource Smoothing Accomplished by Successively Iterating Simulations . . . . .	29
3-1 Graphic Aid Used in Paper Simulation . . . . .	36
3-2 Sample Network . . . . .	61
3-3 Architecture of Computer Implementation of the Decomposition Strategy . . . . .	68
3-4 Demonstration Program Input Processor Logic . . . . .	69
3-5 Scheduling Processor Logic . . . . .	73
3-6 Processing of Commonality Groups Logic . . . . .	84
3-7 Explicit Resource Allocation Logic . . . . .	85
3-8 Sample Flight Network . . . . .	88
3-9 Sample Flight Network With Refurbishment Activities . .	88
3-10 Man/System Interface for Demonstration Scheduling System . . . . .	92
3-11 Timeline Generated Using Resource Modeling of Table 3-2 . . . . .	97
3-12 Timeline Generated Using Resource Modeling of Table 3-4 . . . . .	98
3-13 Timeline Generated Forcing Flight 3 to Occur Before Flight 1 . . . . .	101
3-14 Segmented Graphical Display . . . . .	107
3-15 A Sample Display . . . . .	109
3-16 Minicomputer-related Hardware Components . . . . .	112
3-17 Minicomputer Core Layout . . . . .	113

## TABLES

	<u>Page</u>
2-1 PWV Formulation of the Tire Facility Problem . . . . .	12
3-1 Example Problem Launch Windows . . . . .	94
3-2 Durations and Resources Needed for Example Problem Activities . . . . .	94
3-3 Resources Available for Example Problem . . . . .	95
3-4 Modeling Used for Second Timeliner Execution . . . . .	99
3-5 Specific Resource Allocation for Timeline of Figure 3-13 . . . . .	102

The principal products from the Scheduling Language and Algorithm Development Study are first a programming language and second, an applications program library. Both, of course, have special capabilities appropriate to scheduling and resource allocation problems. The determination of those capabilities has required considerable analysis of a very broad class of problems, and further, a familiarization with problem solving techniques that are being used successfully. Tasks in both Phase I and in Phase II have been aimed at generating or verifying functional requirements for the language and program library. An intended by-product of these tasks has been insight and ideas on how very large scheduling and resource allocation problems can be solved using both computer algorithms and human talents in integrated and organized approaches. These approaches are referred to here as man-computer problem solving strategies.

The purpose of this document is to present the background from which these strategies emerged and to detail the strategies themselves. Emphasis is on the type of scheduling and resource allocation problems that are clearly visible in the planning and activity management of the Space Transportation System (STS). No attempt is made to address explicitly the classes of problems treated in textbooks on operations research methods. These include for example the "traveling salesman" problem, the "knapsack" problem, the "fixed charge" problem, and the "capital budgeting" problem. Yet the problems which have driven the strategy development discussed here are often both sub- and supersets

of these classical problems. For the problem solver who has tried unsuccessfully to conform to one or more of these standard formats, the contents of this volume should be insightful. The thrust is neither to cast real problems into classical formats nor to extend the state of the art in solution methods, but rather to determine how large and complex problems can be solved by a combination of computer algorithms and human skills using iterative and interactive approaches with existing methods.

A major premise underlies the work reported here. It is simply that large and realistically complex scheduling and resource allocation problems can be solved more efficiently and effectively than they are being solved now through the proper allocation of computer capabilities. It is a fact that solutions are being generated manually now, that people know how to produce acceptable schedules and allocations of resources. By understanding the manual methods and the underlying problem solving strategies of capable human schedulers, it should be possible to facilitate strategies that are already proven. From this point of view, the computer can be rejected only on the basis that it encumbers rather than facilitates the solution process or that it attempts to make decisions that the human can make more easily and with higher quality.

Section 2.0 of this report contains data and opinion relating to the use of computer algorithms to produce complete feasible solutions to scheduling and resource allocation problems. Both mathematical programming and heuristic programming techniques are addressed. The



section concludes with a discussion of problem types for which no complete solution can be guaranteed using a single execution of an automated algorithm or algorithm set.

Section 3.0 of this report presents concepts dealing with man-computer problem solving strategies. These strategies are characterized by iterative and interactive use of computer algorithms by a human who participates in the problem-solving process. Data are presented from simulations of two major problem classes that dominate the NASA STS operations and mission scheduling functions. The computer elements of a promising strategy have been implemented and Section 3.0 contains a batch-iterative test case in which a human scheduler employs these programs. Finally, Section 3.0 contains information on the design of an interactive scheduling system and suggests additional investigations that are needed to establish an appropriate man-computer system design.

## 2.0 FULLY COMPUTERIZABLE PROBLEM-SOLVING STRATEGIES

This section discusses the potential of using existing or foreseeable computer algorithms for solving large scheduling and resource allocation problems without the need for human intervention between program input and the final problem solution. Although the distinction between these algorithms discussed in this section and those used in interactive or batch-iterative problem solving strategies is not always clean, the emphasis here is on algorithms that attempt to perform as many of the decision making and associated bookkeeping chores as possible with the user involved in initial input and final output only. Although the recommendations which follow in Section 3.0 result from the deficiency of any algorithm to handle the most general class of large problems, the premise that no problems can be handled completely by existing algorithms is certainly false. The power of mathematical programming and heuristic programming methods is enormous and should be exploited where appropriate. The mathematical or logical details of these algorithms are not discussed here. Rather, the pros and cons of applying them as complete solution strategies are addressed in the following subsections.

### 2.1 MATHEMATICAL PROGRAMMING TECHNIQUES

Mathematical programming algorithms are those which can be shown to produce optimal solutions to problems for which they apply. The guarantee of producing the 'best' solution is an attractive attribute and sufficient to encourage the problem solver to investigate thoroughly the possibility of using an appropriate algorithm from this class. It

is also the motivation for algorithm developers to devise ingenious means for casting problems in the specific formats amenable to one of these methods.

The two subsequent subsections deal first with direct application of mathematical programming methods to entire problems, and second with the application of these methods to mathematically decomposed problems.

#### 2.1.1 Direct Application

Historically, attempts to automate the selection of a mathematical programming algorithm based on problem characteristics have not been completely successful. This is due to the fact that a problem must be modeled, i.e., converted to a computer compatible representation and that this conversion process is not unique. The modeling task is not solely driven from the problem characteristics but is strongly influenced by the solution techniques available. Various techniques are advocated by analysts who have become ingenious in converting problems to the formats compatible to their favored algorithms. Thus the selection of an algorithm usually is influenced by an analyst's background, his available tools, and his style of thinking about problems.

General guides for selecting algorithms can be devised that are at least acceptable to most analysts with broad experience. Such guides are helpful to problem solvers in that they prevent selections that are totally inappropriate or that would result in subsequent reselection. Such a general guideline is shown in Figure 2-1. It is apparent from the figure that dimensionality is a prime factor in selecting a

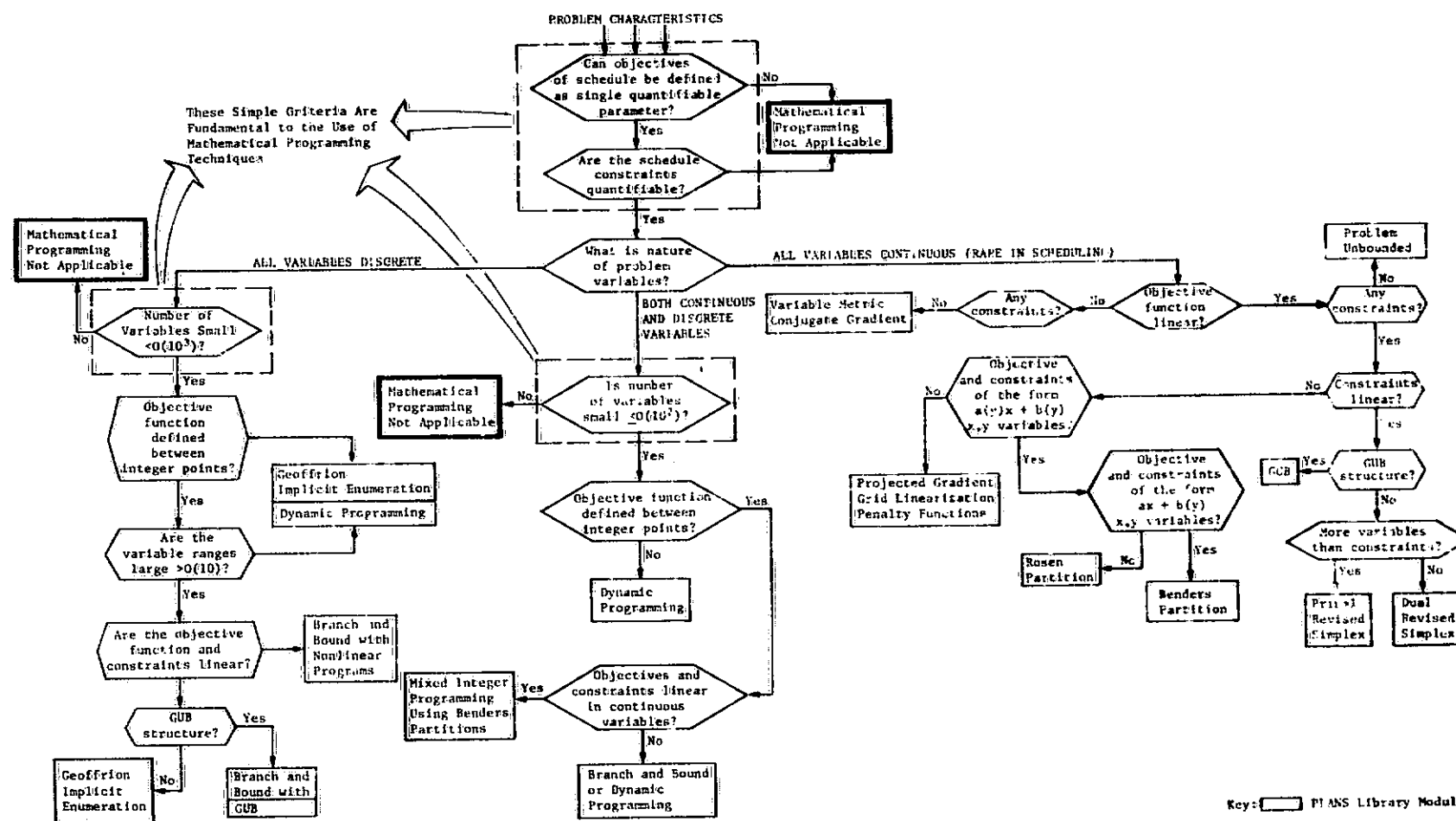


Figure 2-1 Problem Characteristics Amenable to Mathematical Programming

mathematical programming method. Even if different algorithms were substituted at the ends of the decision paths in Figure 2-1, the dominance of the dimensionality constraints would remain. Thus the price for an optimal method is a problem model that possesses 'small' dimensionality. It should be stated clearly that the real problem need not be small, only the mathematical representation of that problem must be within dimensionality limits imposed by the state-of-the-art of mathematical programming methods. For example, if a set of several thousand activities is to be scheduled and the relevant constraints dictate that large numbers of these activities must occur in repetitive and relatively fixed sequences, then the activities in each of these sequences might be merged. The merging would reduce both the number of activities and the number of constraints needed in the problem model and might be sufficient to make mathematical programming an appropriate solution strategy.

A disadvantage of employing modeling ingenuity to make mathematical programming techniques applicable results from the lack of growth potential as the modeling simplification become less and less relevant. Frequently the assumptions that were relevant for the "first solutions" become inappropriate as more sophisticated answers are sought. A scheduling and/or resource allocation system design around mathematical programming techniques could soon be subject to major redesign requirements unless the anticipated problem models are known to be dimensionally founded.

To illustrate this point, consider the problem of grouping payloads into compatible groups for launch with the Space Transportation System (STS). The maximum number of flights per year that the STS can support is known and cannot grow without redesign of the entire transportation system. Similarly, the maximum number of payloads in any flight group is bounded. The grouping problem can be formulated as a set partitioning problem with known maximum dimensionality. Thus, a mathematical programming approach could be considered without undue risk of obsolescence due to growth. Contrast this with a launch ground support activities scheduling problem in which initially only a few activities are defined at a high level. Eventually each of these activities could be broken into entire subnetworks and each of the subnetwork elements could be further subdivided into tasks, etc. Such a problem is one in which growth of the problem model is likely. Thus, even though the initial problem was amenable to a mathematical programming algorithm, the ultimate problem will not be.

Intuitive insight on how large dimensionality is obtained as problems are cast in mathematical form can be gained from the following example problem repeated from Volume 2 of this study's first interim report.

The objective of the problem is to process three different cars through a tire shop in a minimum amount of time. Each car must complete a specified set of jobs in a given sequence (precedence constraints). All the cars must share the same set of tire shop equipment (resource constraints). Additional complexity

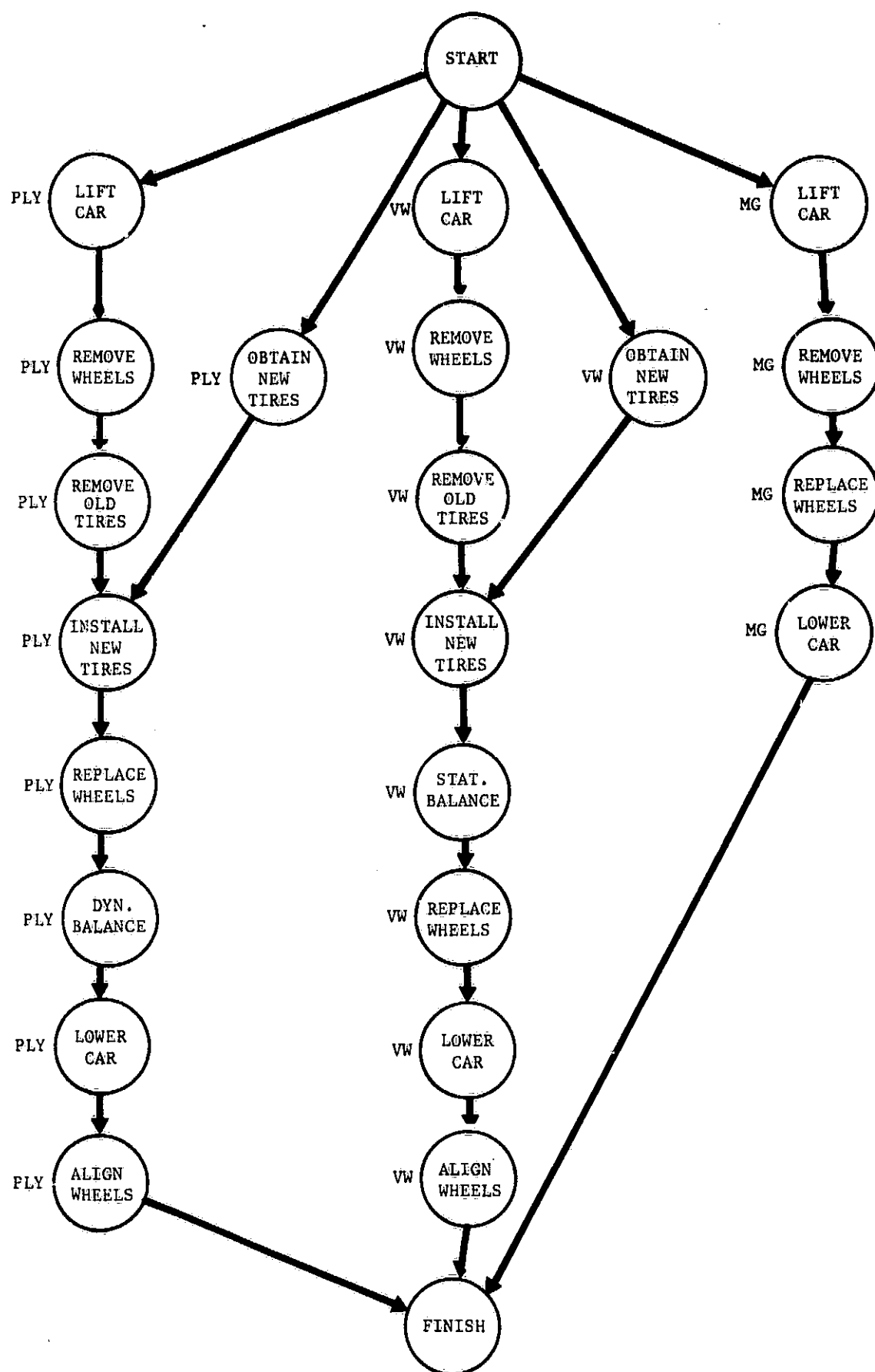


Figure 2-2 Tire Facility Job Network

is added to the problem by the fact that some jobs may be completed by more than one set of resources (substitutability).

The precedence network Figure 2-2 consists of a series of jobs describing the processing needed for each car. The three series are in parallel so that the only interaction between the cars is the sharing of resources.

Problems of this type can be formulated as integer linear programs. However, to keep the resulting program tractable, it is usually necessary to decompose the problem or be somewhat clever in choosing the problem variables. Without taking into account specialties of a given problem, the most efficient integer linear programming (ILP) formulation is that of Pritsker, Watters and Wolfe, (PWW)<sup>1</sup>. Using the PWW formulation, the tire facility problem required 1438 zero-one variables and 474 constraints (see Table 2-1), while the largest single-level integer linear programs currently being solved handle less than 300 variables. Even if the problem were decomposed (processing each car would be a subproblem), the dimensionality is excessive.

The majority of the variables result from the high degree of resource substitutability. The PWW formulation requires each substitutable combination to be treated as a separate job with a constraint added to insure that only one of the combinations appears in the solution. If each job used only one set of resource types,

---

<sup>1</sup> Multi-project Scheduling with Resources: A Zero-one Programming Approach. Management Science, Vol 16, No 1, September 1969.



Table 2-1. PWW Formulation of the Tire Facility Problem

Two cases considered:

- 1) Nominal case - unique resources for each job;
- 2) Substitutable case - more than one set of resource types may be used for each job.

Comment: In PWW formulation, substitutability is handled by creating a new job for each different feasible combination of resource types that can be used on a given job.

Number of jobs in network:

nominal	22
substitutable	95

Number of 0-1 variables:

$$= \sum_{\text{jobs}} (\text{slack periods} - 1)$$

nominal	282
substitutable	1438

Number of constraints:

- 1) Precedence constraints - number of jobs with predecessors

nominal	17
substitutable	94

- 2) Substitutability constraints = number of substitutable jobs

nominal	0
substitutable	20

- 3) Resource sharing constraints

$$= \sum_{\text{Resources}} (\text{possible sharing periods})$$

nominal	285
substitutable	285

- 4) Objective function constraint

nominal	1
substitutable	1

- 5) Single job completion time constraints

nominal	22
substitutable	95

the problem could be formulated using 282 variables and 324 constraints. Still, the tire changing problem is small and relatively simple (only 22 jobs). More realistic problems requiring a computerized solution are large enough to be completely unsolvable using an ILP formulation.

The principle reasons for the high dimensionality of the tire changing problem are the large number of time periods necessary to describe the precedence constraints and the large number of different resource types. If temporal constraints are to be described, the number of time intervals and consequently the number of problem variables usually increases. Similarly, the number of problem variables is directly related to the number of resource types needed for each job.

Reference to Figure 2-1 indicates that the application of mathematical programming techniques also requires that a single quantified objective and quantified constraints be defined. This requirement can be helpful in causing agreement to be reached on the real purpose for solving a problem, or it can be harmful because it imposes artificialities on the problem model that are not there in the real problem. The case for explicit quantifiable constraints is stated effectively by Forrester<sup>1</sup>.

The mental model is fuzzy. It is incomplete. It is imprecisely stated. Furthermore, within one individual, a mental model changes with time and even during the flow of a single conversation.

---

<sup>1</sup> Jay W. Forrester: Testimony for the Subcommittee on Urban Growth of the Committee on Banking and Currency U.S. House of Representatives, October 7, 1970. Copyright 1971.

The human mind assembles a few relationships to fit the context of a discussion. As the subject shifts so does the model. When only a single topic is being discussed, each participant in a conversation employs a different mental model to interpret the subject. Fundamental assumptions differ but are never brought into the open. Goals are different and are left unstated. It is little wonder that compromise takes so long, and it is not surprising that consensus leads to laws and programs that fail in their objectives or produce new difficulties greater than those that have been relieved.

For these reasons we stress the importance of being explicit about assumptions and interrelating them in a computer model. Any concept or assumption that can be clearly described in words can be incorporated in a computer model. When done, the ideas become clear. Assumptions are exposed so they may be discussed and debated.

If the debate on the assumptions referred to by Forrester is more difficult to resolve than solving the original scheduling problem itself, the requirement for a single quantifiable objective is a deterrent to the choice of mathematical programming methods. Typical of this situation is the almost endless search for 'weighting' factors. Typical questions that arise are "What is the equivalent benefit in dollars of an equal distribution of payload types flown over a one year period?" or "What is tradeoff between risk of being able to hold to an intricate schedule with little slack as opposed to the dollars saved if such a schedule is followed successfully?"

An objective view of the requirement for a single quantifiable objective and quantifiable constraints would admit that the resulting problems are not unique to mathematical programming methods. They occur whenever computer decision-making is employed. It is safe to state, however, that the rigorous formats required by a mathematical program encourage greater use of artificial constants, normalizing factors, and benefit ratios than do other classes of solution algorithms.

#### 2.1.2 Application via Mathematical Decomposition

The dimensionality limitations imposed by the direct application of mathematical programming techniques are relieved substantially if the problem model lends itself to mathematical decomposition. Decomposition is the partitioning of a problem into smaller subproblems that can be solved independently but which are related mathematically. Because of the mathematical relationships that exist between subproblems, an overall optimal solution can be obtained by iterative solutions of the subproblems.

Methods of decomposition are varied and numerous. Books by Wismer<sup>1</sup> and Lasdon<sup>2</sup> are excellent references for the mathematical details of available decomposition strategies. Several examples related to scheduling are provided in Volume 2, First Interim Report, Scheduling Language and Algorithm Development Study, February 1974.

---

<sup>1</sup> Wismer, David A., Ed., Optimization Methods for Large-Scale Systems ... with applications, McGraw-Hill Book Company, 1971.

<sup>2</sup> Lasdon, Leon S., Optimization Theory for Large Systems, The Macmillan Company, 1970.

Decomposition techniques have extended the utility of mathematical programming techniques substantially. Problems that are well suited for the direct application of mathematical programming methods may often be substantially enlarged and solved through the use of decomposition. However, the class of problems that are mathematically decomposable is a subset of those that can be cast in the appropriate format for a direct application of mathematical programming. Thus the effort of transforming the problem description to a mathematical model must be done before it can be recognized whether decomposition factors are appropriate. For example, a decomposition strategy called Generalized Upper Bounding is only applicable to problems that can be expressed as linear programs and which in addition have a particular tableau structure (see Lasdon, 1970, page 324, op. cit.). Thus, if a problem exists that is already expressible as a linear program and is pressing the dimensionality limits of that method, some effort is worthwhile in checking for particular structures amenable to decomposition.

However, because not all problems are expressible in forms amenable to mathematical programming, and because those that are cannot always be decomposed, the use of optimizing methods is still restricted rather severely. Even when decomposition is possible, the resulting subproblems may themselves be too large. Finally, the practicality of computer execution time must be considered when many iterations through large subproblems is required for a single converged solution.

In those cases where either mathematical or practical limitations appear inevitable, the advantages of guaranteed optimality must be

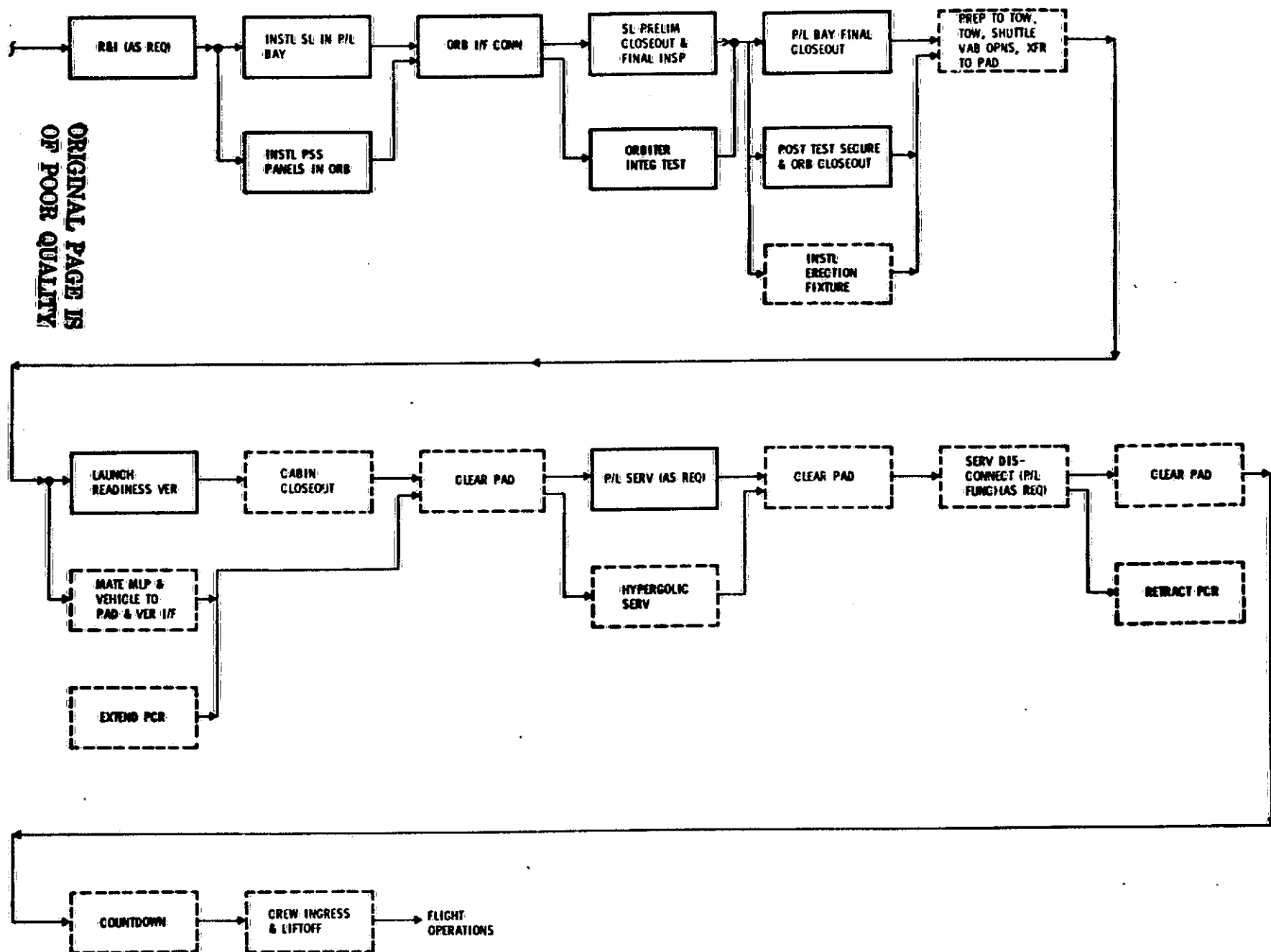
sacrificed. Algorithms that produce solutions that satisfy all constraints and that approach the optimal objective are called heuristic methods. The use of heuristic methods for completely solving scheduling and resource allocation problems is discussed in the next section.

## 2.2 HEURISTIC ALGORITHMS

Heuristic logic offers increased flexibility in modeling various problem characteristics. Any method that produces a feasible solution but which may not produce the optimal solution is considered a valid heuristic method. Thus, the variety of methods is unlimited; often heuristic solution methods are devised for a single problem and customized to that problem's peculiar characteristics. Thus it is impossible to construct a rigorous classification of heuristic methods. However, an informal correlation between problem characteristics and heuristic characteristics can be made and is used for organizational purposes in this section.

### 2.2.1 Problems With Dominating Network Constraints

Scheduling problems often deal with activities that are related to one another by relationships of sequences. Such problems are usually described with network or task flow diagrams. Figure 2-3 illustrates a portion of a task flow diagram associated with Spacelab operations. If durations are attached to each element of such a network diagram, and if the connections in the network indicate only predecessor/successor relationships, a schedule can be generated by using earliest start time decision criteria for each task. The well known PERT and Critical Path Methods (CPM) are relevant to this class



of problems. A top level logic diagram of time-progressive heuristics (as a class) is shown in Figure 2-4. The nature of the sequence relationship between jobs can be generalized to include more realistic relationships. For example, if JOB B must start within three hours of when JOB A ends, the relationship between JOB A and JOB B is no longer a simple predecessor. In this document, job relationships that are not simple predecessor/successor relationships are called general temporal relations. Standard network analysis techniques have to be extended to accommodate such relations. The extensions are a good example of how heuristic methods evolve. As new problem characteristics must be accommodated by the solution algorithm, appropriate logic is added. Eventually the logic may be a good representation of a skilled analyst's solution strategy, yet at the same time it may be limited in its generality.

In problems where the constraints include temporal relations between all jobs (i.e., a network defines the operations) the algorithms that are applicable are those which step through time segments in increasing or decreasing order. Considering each time interval in turn, the algorithms assign start times for those jobs that can be started in that interval. Often resource constraints must be considered in the algorithm to determine if jobs can be scheduled in each time frame. The characteristic of considering time segments in sequence provides the term "time-progressive" for this class of algorithm. It is logical that time-progressive logic is appropriate for problems with dominant temporal relations since any logic should attempt to satisfy the most



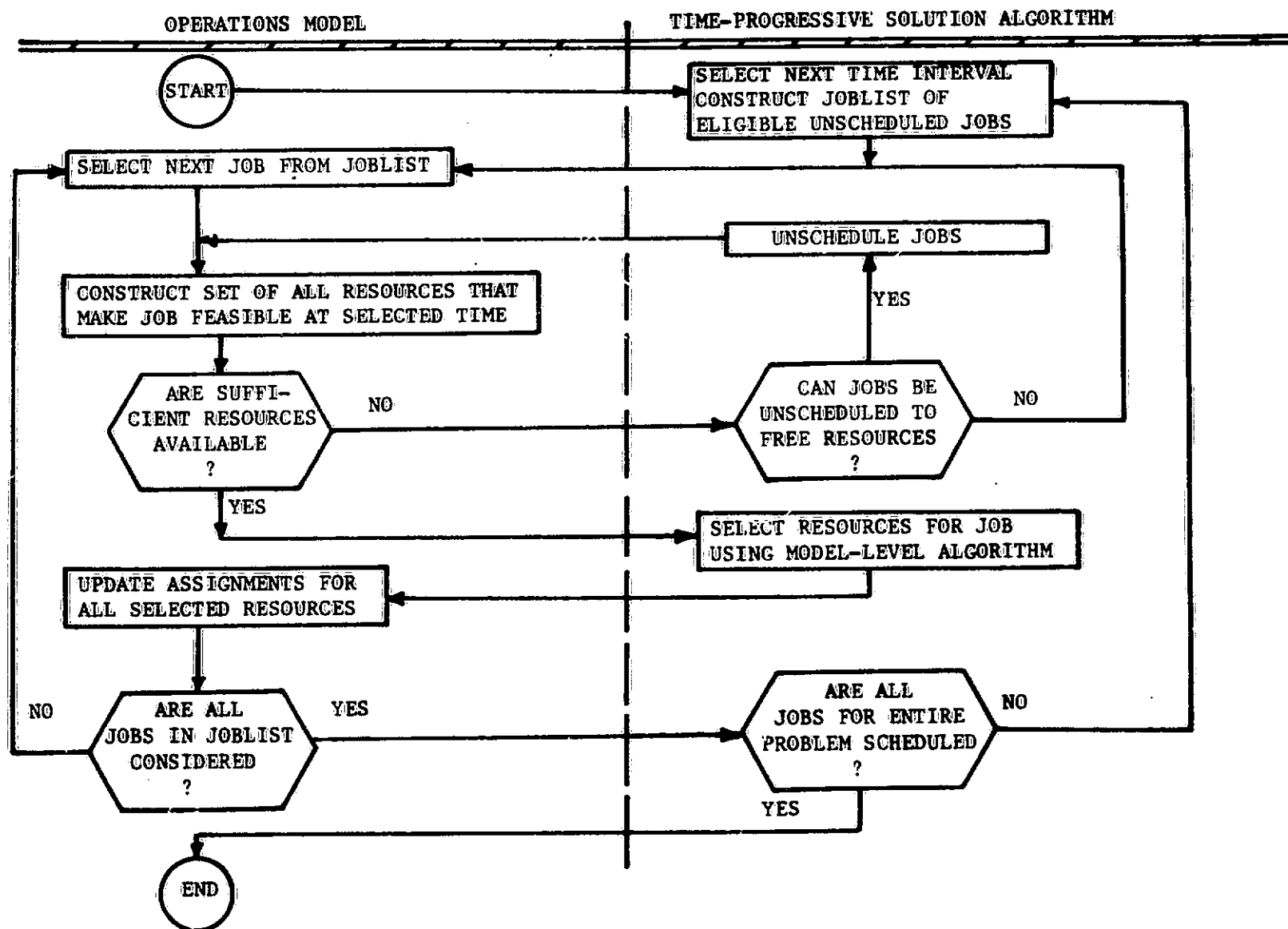


Figure 2-4 Top Level Logic Diagram of Time-Progressive Heuristics

restrictive constraints first. In the case of network problems, the dominating constraints involve time sequencing. Therefore, appropriate solution algorithms proceed in a time-sequential fashion.

Problems dominated by network constraints or general temporal relations are usually minimum time problems, i.e., usually include the objective of completing all jobs in the shortest span time. Modification of this objective might include the restriction that maximum resource demands be limited or even "smoothed". Often the sensitivity of the span time to complete all jobs to a delay or extension in any one job is of interest. Critical path analyses can be performed on network problems in order to determine such sensitivities. Slack is a measure of how much schedule shippage can be allowed in the start or finish of any one job before the entire span time for completing all jobs is affected. Computation of slack also requires a time-progressive analysis.

Thus a correlation exists between problems with dominating network or temporal relations and a type of heuristic rule called a time-progressive algorithm. Even this weak correlation is not complete, however, since some problems with network constraints have very strong resource constraints. In such cases time-progressive logic may not be efficient. Problems without dominating network or temporal relations are discussed in the following subsection.

#### 2.2.2 Problems Without Dominating Network Constraints

For the sake of brevity, problems in which network constraints are missing or at least non-dominant will be called activity scheduling

problems. Usually such problems have the objective of fitting as many activities into a fixed time period or window as possible. The constraints are normally the availability of resources to support each job or alternatively the availability of resources in the correct state. In activity scheduling problems, it is usually not important which order the jobs are scheduled. If some jobs must precede others it is usually because the predecessor jobs produce resources that the successor jobs need. Therefore, temporal relations can often be removed from the problem model by forcing compliance with sequence relations through the use of appropriate required resources for each job.

An example of an activity scheduling problem from the NASA context is experiment scheduling in an earth orbital mission. Experiments that must be scheduled in short windows (e.g., over certain points on the earth) or experiments that are critical to mission success provide the dominating constraints. These experiments should be scheduled before the less critical or more flexible experiments are considered.

Heuristic methods for activity scheduling problems are usually based on a priority list of jobs. The jobs are scheduled in order from the priority list and are given start times for 'best fit' to the partially completed schedule. The start times are usually not assignable in monotonic order, thus the name given to heuristic algorithms appropriate for activity scheduling problems is 'time-transcendent.' A top level logic flow diagram of time-transcendent heuristic (as a class) is shown in Figure 2-5. The decision criteria within time-

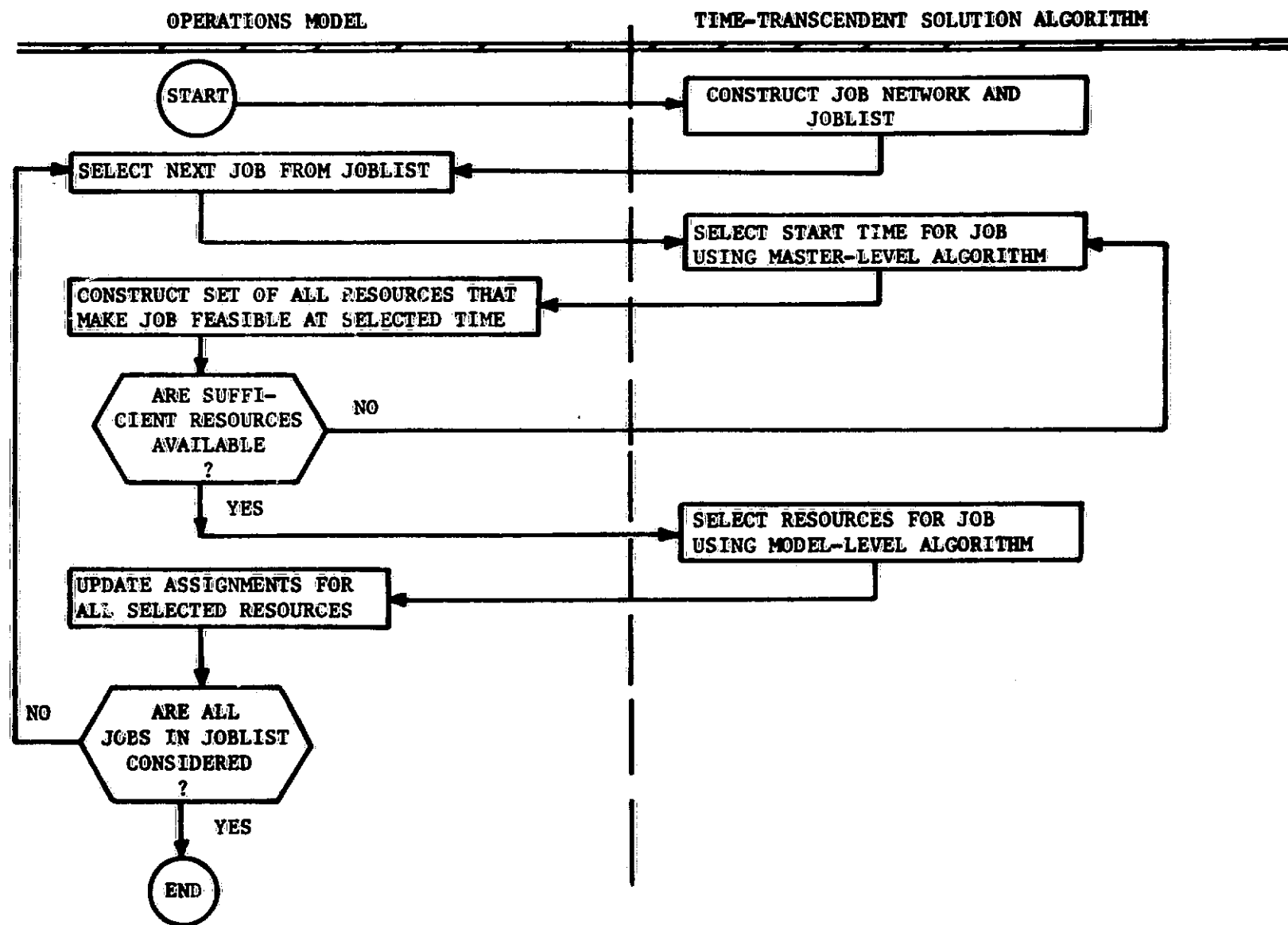


Figure 2-5 Top Level Logic Diagram of Time-Transcendent Heuristics

transcendent heuristics are extremely customized. Intuitive measures of scheduling flexibility are derived by almost every different analyst. For example, one might arbitrarily assign a priority number to each job and let the computerized heuristic assign highest priorities first, etc. Alternately, the priorities could be calculated by the computer using some measure of time slack within a window plus percentage of total resources used. The priority ranking may be computed only once, or may be recomputed after every N assignment (where N may vary from 1 to any integer). The variations on this type of logic are infinite; therefore, time-transcendent scheduling programs are usually highly specific to a single application.

Often activity scheduling problems are overconstrained. There may be no solution that assigns all jobs a start time and also violates no constraints. The desired solution may be to schedule as many jobs as possible without violating any constraints or to assign all jobs but to violate the constraints as little as possible. This latter objective is particularly difficult to realize in a computer program. Soft constraints can be handled only if a measure of hardness is associated with each constraint. Usually it is difficult or impossible to quantify the comparative importance of constraint violation. For example the algorithm designer might have to decide if scheduling a sleep period for a crewman to start thirty minutes late was more or less undesirable than using 110% of the recommended RMS power for two minutes. The subjectivity of such decisions leads to extreme difficulty in designing computerized logic. The need to permit the human to make such decisions

is discussed subsequently. However, time-transcendent heuristic algorithms that incorporate logic to handle choices between constraints to violate or decisions about constraint softness are so specialized that they have very limited general applicability.

Some standardized heuristic algorithms do exist, however, and these are the subject of the next subsection.

### 2.2.3 Integrated Heuristic Programs for Large Problems

This subsection discusses briefly four categories of general heuristic algorithms that are available in preprogrammed form and which are relevant to large scheduling problems. Since references to most of these techniques have already been made, the emphasis here is on functional limitations of the technique.

The first generally available class of heuristic algorithms are the network analysis routines. These routines can produce schedules for problems that contain only predecessor or successor constraints. Job durations are required also. Extension to network analysis methods permits the automatic decomposition of large networks into subnetworks that contain all their own predecessors and successors, the conversion of job networks into event node networks (events have no duration but are separated by delays), the assembly of separate networks that contain common events, the incorporation of earliest and latest start times for each job (windows), and the incorporation of continuous successor relations (JOB B must start when JOB A ends). The PLANS module library (Volume III, Phase II Final Report) contains a complete set of extended network scheduling algorithms.

The deficiencies of these algorithms include the following items:

1. no resource relations are handled;
2. only a subset of problems described in Subsection 2.2.1 are handled;
3. only a limited class of temporal relations are handled.

The well known PERT and Critical Path methods are basically network analysis algorithms. These techniques have been extended to permit estimates of durations to be given by the problem solver and, in some cases, to permit resource requirements for each job to be considered. Although the inclusion of resources into a PERT analysis is sometimes called Extended PERT, it more commonly is called Project scheduling.

Project scheduling algorithms allow problem resources to be modeled as pools of identical items that are substitutable one for another within a single pool. For example, a manpower pool contains a certain number of persons with a single skill, any or all jobs to be scheduled may required a certain number of persons with this skill. When a job is scheduled the pool is depleted of the number of persons required by the job unless that number is not available. Because the power of network analysis is coupled with a resource model, project scheduling algorithms are very useful. The problem model (networks plus resource pools) is simple enough that logic for high dimensional problems can be written that executes efficiently. The PLANS library contains a complete set of project capabilities for smoothing resource pool usage, and for optional usage of contingency resources if necessary to avoid an extension of the total project duration. The functional deficiencies of project scheduling algorithms are:

1. only predecessor/successor temporal relations are handled;
2. only pooled resources are handled; i.e., assignment of specific resource elements cannot be done.

Discrete simulation methods are available which can be used to solve scheduling problems. GPSS, SIMSCRIPT, GASP, SIMPL/I, SIMPL, and others are effective tools for discrete simulation. The similarity of simulation logic to time-progressive heuristics is apparent. To understand the functional deficiencies of discrete simulation for solving scheduling problems, one has to consider the distinctions between the information that is sought when performing a discrete simulation and that which is sought when solving a scheduling problem.

Typically, discrete simulation analysis seeks to find where delays occur, or where queues build when various loads and/or contingencies occur. The operational policies and facilities are chosen and then subjected to hypothetical operational conditions. The objectives of the simulation analysis are usually to configure the operational system in a manner to handle expected real environments in an optimum way. Before each simulation, decision rules (usually queueing policies) are established. Usually the desired output is a statistical measure of operational performance that can be expected from a statistically modeled demand profile and/or contingency event profiles. Much of the power of discrete simulation tools is derived from automated statistics gathering and random event generation capabilities.

Scheduling, by way of contrast, involves a set of defined operations with known (i.e., not statistically modeled) input traffic. The



objective of solving a scheduling problem is to find decisions that will utilize the operations facilities and resources in an optimum way for single set of tasks. The table below emphasizes the contrast between scheduling and discrete simulation.

Simulation

Decision criteria for operational alternatives are pre-specified.

Jobs to occur and event contingencies are usually modeled statistically.

Objective often is to modify operation procedures to maximize throughput efficiency or quantity over a statistically described range of requirements.

Scheduling

Decision criteria for operational alternatives are the output objectives.

Jobs to occur are known and described explicitly.

Objective is to maximize throughput efficiency or quantity for a known set of requirements and explicitly defined operational facilities and policies.

As stated in previous subsections, the satisfaction of scheduling objectives may require a time-transcendent logic which is not possible using discrete simulation tools. However, for problems amenable to time-progressive logic, discrete simulation tools used without the Monte Carlo Statistical approach and use in an iterative manner can produce good schedules. Figure 2-6 shows a set of simulations in which the resource availability level was successively reduced between runs in order to effect a smooth resource profile. Since the scheduling problem was of the network-dominant type (Subsection 2.2.1) the resulting schedule is comparable to that which would have been produced by a time-progressive scheduling algorithm.

An appreciation for the subtle difference between scheduling and simulation objectives can be gained through a simple example. Suppose

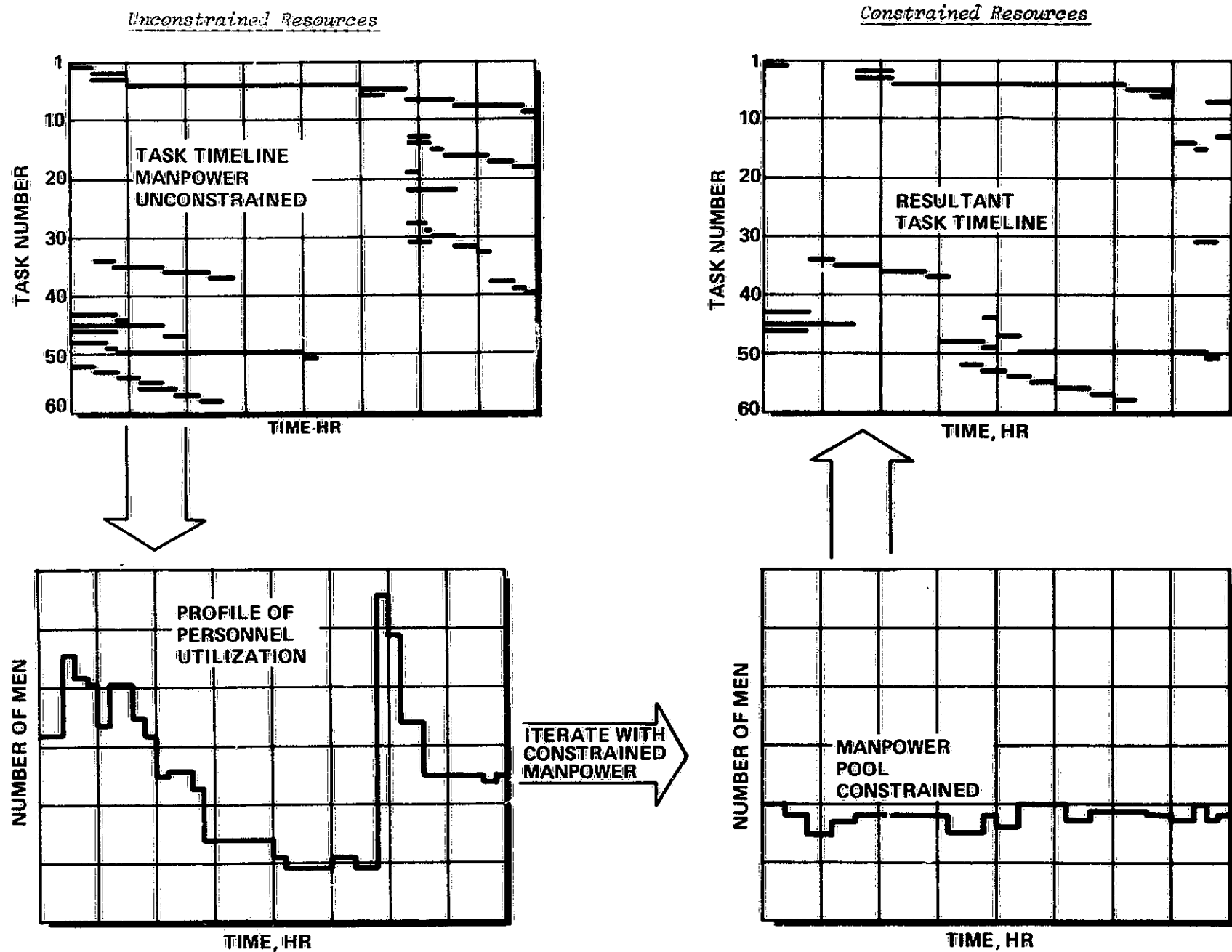
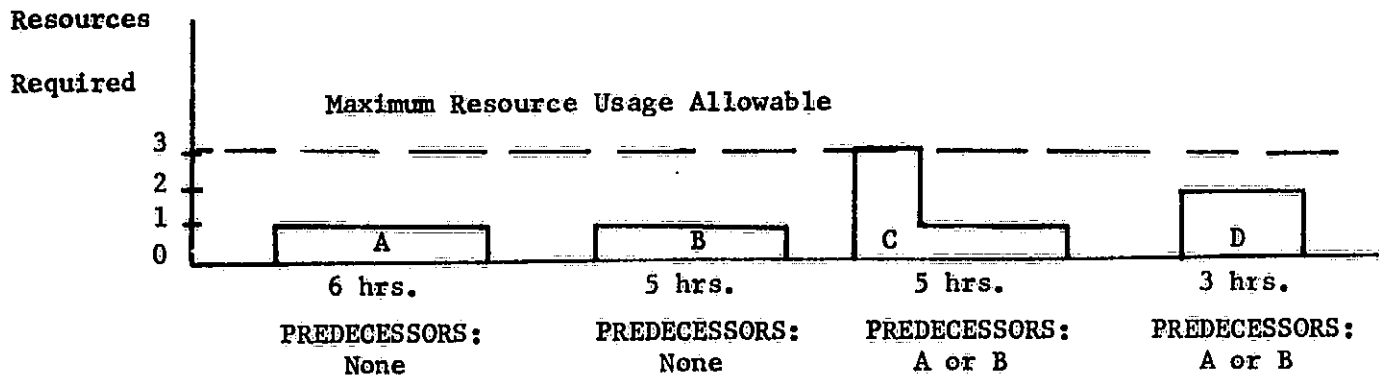
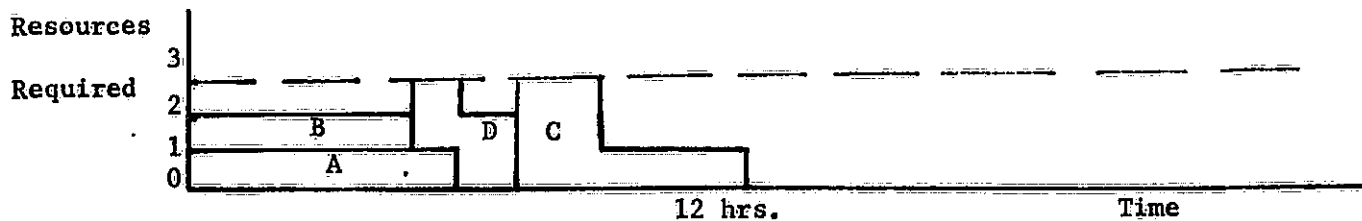


Figure 2-6 Resource Smoothing Accomplished by Successively Iterating Simulations

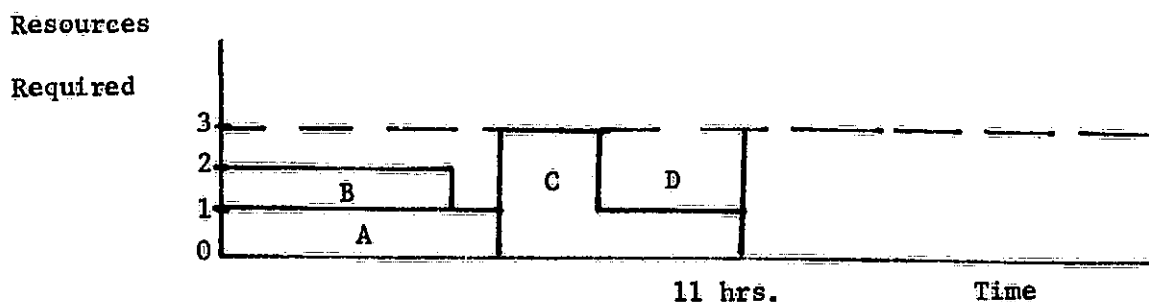
four jobs were to be scheduled; each job had the duration and levels of required resources shown in the figure below:



A simulation which used the simple decision rule of starting each job as soon as possible would build the following schedule:



A minimum time scheduling algorithm should discover, however, that by not scheduling either job C or job D when job B ends but rather waiting one hour until job A ends produces a shorter schedule:



To produce the proper schedule, the user of the simulation code would have to recognize that a reordering of the start times for jobs C and D would improve the output and modify the input by some artificial means. He could create a dummy job or perhaps program a look-ahead decision rule. The look-ahead rule is really logic typical of scheduling and is difficult to program using simulation tools because of the necessity to run the simulated clock forward and then backwards again. Thus although simulation tools can be helpful, they are not ideally suited nor completely adequate for solving scheduling problems.

### 2.3 THE PROBLEM MODEL - ALGORITHM GAP

Neither mathematical programming techniques nor generalized heuristic methods provide all the capabilities needed to solve the scheduling and resource allocation problems that are typically associated with shuttle operations. The operations models are generally larger than mathematical programming methods can handle, or more complex than heuristic methods can handle. Although decomposition methods and greater computing efficiency are permitting larger and larger problems to be solved with MP's, the dimensionality gap appears too great to expect that current scheduling systems could realistically be designed solely around such methods. Historically attempts to modify heuristic logic to accommodate more complex problem models have led to very complex programs that are very difficult to debug and which are extremely sensitive to small changes in problem scope.

Nearly two years of searching for fully automated or automatable methods to solve the scheduling and resource allocation problems

foreseen in shuttle-related applications has led to the conviction that a set of algorithms is only a part of the answer. Even with a programming language such as PLANS to permit easy development and modification of programs employing the set of algorithms, the solution of the complex problems will require iterative approaches. With this realization in mind, analyses of man-computer iterative and interactive solution strategies were undertaken. The hope was that smaller and logically less complex algorithms could be linked into a logical process involving man's ability to make decisions based on subjective information, recognized pattern and a conceptual understanding of constraints and objectives. The results of these investigations have been very encouraging. They are reported in the following sections.

### 3.0 HUMAN PARTICIPATION IN COMPLEX SCHEDULING TASKS

If the human capabilities are to be exploited in order to provide solutions to scheduling and resource allocation tasks that defy entirely automated solution, then a fundamental problem is the proper allocation of roles for man and computer. This apparently straight-forward task is evidently not so simple since user-defined systems have not been as successful as might be imagined. Noncritical acceptance of user requirements as stated by the user appears to be the first step on the road to potential disaster. Most users understand the end product they are to deliver (e.g., a schedule or resource profile) but very few users fully understand and can express verbally or mathematically the process that they use to perform that job. Yet an understanding of the total job to be performed by the total man-computer system is necessary before it is possible to appropriately allocate tasks to man and computer.

What is needed then, are formal methods for developing an understanding of the total problem solving process. Such formal methods are scarce today; research related to this problem but beyond the scope of this study is being conducted to develop and improve such methods. The sections which follow describe materials that were used in this study to determine typical user's logical processes in solving scheduling problems.

#### 3.1 METHODS FOR ALLOCATING ROLES IN MAN-COMPUTER SYSTEMS

Proper allocation of roles demands a complete understanding of the total task to be performed by the man-computer system. One method for determining this total task is to simulate the problem solving environment

with test subjects as problem solvers who have at their disposal hypothetical but functionally described problem solving aids. This technique is referred to here as 'paper simulation'. In the context of this study, the problem solving aids are computer programs. A real implementation of these programs is not required to conduct a paper simulation. However, the functional logic that exists between input and output must be precisely describable so that a 'manual' execution can be accomplished in support of the test subjects usage of the 'program.'

The two paper simulations described here led to important conclusions. In the first case, several functional capabilities were discovered that are not now supported directly by the PLANS program library (see Volume III of this report). In the second case, a postulated problem solving strategy for a major class of problems was supported. This has led to a batch iterative program design and later to an interactive scheduling system design using a vector graphics terminal as an interface device (Section 3.5).

The second paper simulation also supported a postulated model of user decision behavior that had been formulated. The ability to construct such a model was encouraging because it provided a concise and surprisingly complete description of user logical processes. If decision modeling could be developed to a formalized procedure it would be a major contribution to the proper design of man-computer systems.

## 3.2 PAPER SIMULATION OF ACTIVITY SCHEDULING PROBLEM

### 3.2.1 Introduction

A paper simulation was performed to study the activity scheduling problem. Five people participated - two experienced activity schedulers, two experienced project schedulers and one naive scheduler. Each subject was given a statement of the problem to be solved (see Section 3.2.2). Four participants requested a graphic aid which was provided as shown in Figure 3-1. All participants were requested to record the way they solved the problem, specifically the order in which they placed activities on the timeline, the information they needed to solve the problem and any regressions in their logical approach.

The problem was conceived as a typical activity scheduling problem facing flight planners on a manned earth orbiting mission. The different 'experiments' were designed to illustrate typical operating constraints present on these types of missions in order to detect how the participants solved problems of this type.

### 3.2.2 The Activity Scheduling Problem Statement

Following is a list of experiments and other activities which must be scheduled during a two day mission for two crew members. This mission is on an orbiting spacecraft and many of the experiments are dependent upon orbital parameters for their data acquisition. Thus, in addition to experiment and activity descriptions, the scheduler will be provided with a listing of orbital parameters necessary to perform the task.



1	
2	

[illegible]

**Figure 3-1 Graphic Aid Used in Paper Simulation**

All activities, experiments and other performances alike, must be scheduled in a manner consistent with their individual specified constraints in order to satisfy the mission goals.

#### Additional Requests

1. Please write down the order in which you put the schedule units on the timeline.
2. What is the most significant piece of information needed to make each assignment to the timeline?
3. What aids would you like in solving this problem?
4. When you've finished, what kind of feedback would you expect from an analytical source?

#### Experiment Descriptions

- A. Duration 1.6 hrs.

Must be performed on each crew member at  $24 \pm 2$  hr. intervals.

It must be performed no less than 1.5 hrs. after a meal.

- B. Duration .3 hr.

Requires one crew member. It should be performed as often as possible with a maximum of 1.5 hrs. per day.

- C. Duration .75 hr.

Must be performed immediately after every meal for each subject.

- D. Duration 1.3 hrs.

Must be performed three times during the mission. Only one crew member is required to perform it, but it uses the only piece of equipment of type A on board.

E. Duration .4 hr.

Must be performed as often as possible for a minimum of three non-concurrent operations a day. Each performance must start .1 hr. before orbital sunrise. It requires only one crew member for operation.

F. Duration .25 hr.

Must be performed once each day on each crew member while he is eating a meal.

G. Duration 2.0 hrs.

Requires one crew member and must be performed once per mission. It cannot be performed while experiment E is being performed.

H. Duration .75 hr.

Requires one crew member and must be performed once per mission. It must be started during the night cycle.

I. Duration 2.2 hrs.

It must be performed when daylight sighting of target is in view. The performance requires a one hour set up and a one hour stow with the .2 hr. centered over the target sighting. Requires both crew members for the set up and performance, but only one for the stow. It must be performed as often as possible.

J. Duration 1.5 hrs.

Requires both crew members. Must be performed once per mission.

K. Duration 1.6 hrs.

Must be performed once per mission. Requires one crew member. Requires only piece of equipment of type A on board.

All of the above experiments should be scheduled according to their constraints. However, experiments A, C, E and I are especially important in that their completion goes the furthest towards satisfying the goals of this mission. The data received for these particular experiments will be degraded if their constraints are not met. In particular, I should be performed at every feasible opportunity.

In addition to the above experiments, meals must be scheduled three times a day and it is desirable that the crew eat together. One hour must be allotted for each meal.

Meal #1 must start at 0100  $\pm$  30 min.

Meal #2 must start at 0600  $\pm$  60 min.

Meal #3 must start at 1200  $\pm$  30 min.

Each crew member should have a minimum of 2.5 hours a day of personal time which also must be included in the schedule. The minimum schedule period for this activity is .25 hours. Each crew member must have some personal time immediately after rising and it's desirable that each one has some time immediately before the sleep period. It is mandatory that they have some time to themselves in the forenoon and the afternoon.

There are two crew members on this mission.

The crew must have their sleep to insure the success of the mission and thus the sleep period for both crew members (1600-2400 hours day 1 and 400-4800 hours day 2) is sacrosanct.

The schedule interval is 48 hours long with each day 24 hours long and the start of the first day is 0000 hours. A military clock is used.

# DAY/NIGHT CYCLES DAY 1

ORBIT #	NIGHT CYCLE		DAY CYCLE	
	START	STOP	START	STOP
1	0000	- 0030	0030	- 0130
2	0130	- 0200	0200	- 0300
3	0300	- 0330	0330	- 0430
4	0430	- 0500	0500	- 0600
5	0600	- 0630	0630	- 0730
6	0730	- 0800	0800	- 0900
7	0900	- 0930	0930	- 1030
8	1030	- 1100	1100	- 1200
9	1200	- 1230	1230	- 1330
10	1330	- 1400	1400	- 1500
11	1500	- 1530	1530	- 1630
12	1630	- 1700	1700	- 1700
13	1800	- 1830	1830	- 1930
14	1930	- 2000	2000	- 2100
15	2100	- 2130	2130	- 2230
16	2230	- 2300	2300	- 2400

## TARGET COVERAGE - EXPERIMENT I DAY 1

0524	-	0536
1224	-	1236
1901	-	1913

# DAY/NIGHT CYCLES DAY 2

ORBIT #	NIGHT CYCLE		DAY CYCLE	
	START	STOP	START	STOP
17	2400	- 2430	2430	- 2530
18	2530	- 2600	2600	- 2700
19	2700	- 2730	2730	- 2830
20	2830	- 2900	2900	- 3000
21	3000	- 3030	3030	- 3130
22	3130	- 3200	3200	- 3300
23	3300	- 3330	3330	- 3430
24	3430	- 3500	3500	- 3600
25	3600	- 3630	3630	- 3730
26	3730	- 3800	3800	- 3900
27	3900	- 3930	3930	- 4030
28	4030	- 4100	4100	- 4200
29	4200	- 4230	4230	- 4330
30	4330	- 4400	4400	- 4500
31	4500	- 4530	4530	- 4630
32	4630	- 4700	4700	- 4800

## TARGET COVERAGE - EXPERIMENT I DAY 2

2538 - 2550  
3224 - 3236  
3912 - 3924

### 3.2.3 Results

The results of the simulation indicated that an interactive system with graphic aids used iteratively was an effective way to solve an activity scheduling problem. It is most efficient to assign the computer to the mechanical bookkeeping tasks such as placing activities on the timeline until a conflict occurs and have the human perform the more complicated tasks of assessing which activities need to be re-scheduled, or which constraints need to be relaxed in order to resolve the conflict. The functional capabilities required for solving the activity scheduling problem are basically window orderers, window finders, window filters and window fillers where a window is a time period during which no activities are yet scheduled.

A window finder would provide the scheduler with the capability to determine all of the scheduling opportunities, or windows, available on the timeline for a certain activity based upon its unique operating constraints and requirements. For example, a window finder might be asked to find all windows on the timeline greater than four hours in duration, during which a plumber is available. A window filter provides the scheduler with the capability to filter available scheduling windows on the timeline based on criteria which he specifies, such as eliminating those windows above which occur after 5:30 PM. A window orderer orders available scheduling windows by criteria specified by the user, such as shortest duration windows to longest duration windows. A window filler would provide the scheduler with the capability to automatically fill all scheduling windows, satisfying certain criteria

with a specified activity. For example, the scheduler might want to fill all windows of duration of 15 minutes or less with routine maintenance.

The functional capabilities just described, together with an interactive system used in an iterative mode would greatly simplify the complicated task of solving activity scheduling problems.

### 3.3 PAPER SIMULATION OF PROJECT SCHEDULING PROBLEM

#### 3.3.1 Introduction

The object of the simulation was to see how difficult or easy it was to use a particular decomposition solution strategy as the basis for a scheduling system. A prototype system was defined consisting of two automated modules and a set of instructions. A problem was provided. The problem could have many solutions. A solution consisted of start times for all jobs needed to launch three shuttle flights and the specific resources needed to accomplish each job.

The problem statement and the other information provided to the simulation subjects are presented here to provide the context for the conclusions and recommendations which resulted.

#### 3.3.2 The Project Scheduling Problem Statement

##### Problem

Assume that the space transportation system is in full operation. Approximately 50 flights a year are being flown using a fleet of boosters, orbiters, etc. The task is that of scheduling 3 flights which must be launched between time = 0 and time = 35. To launch these flights, the following resources are to be used; they can be assumed to be ready



to use (i.e., all items which need refurbishing will have been refurbished) at time = 0:

Orbiters #135, #806

Launch Pad #7

Solid Rocket Boosters (SRB's) #511, #512, #513, #514, #515, #516

External Tanks (ET's) #201, #213, #227

Integration Cells (IC's) #3, #6

Each launch requires the following activities:

Prepare Orbiter

Prepare Booster

Mate Orbiter and Booster

Launch

Additionally, orbiters and pads must be refurbished after each use before they can be prepared for another flight.

"Mate Orbiter and Booster" cannot start until both the orbiter and booster have been prepared.

Each flight must be launched during its launch window.

	Launch Window Start Time	Launch Window End Time
Flight 1	13	35
Flight 2	15	23
Flight 3	19	24

Information about the requirements for each of the launch-related activities (durations are in days).

### Activity Definitions

#### Prepare Orbiter

Duration - 4

Resource Needed:

1 Orbiter

#### Prepare Booster

Duration - 9

Resources Needed:

1 External Tank

2 SRB's

1 Integration Cell

#### Mate Orbiter and Booster

Duration - 4

Resources Needed:

1 External Tank

2 SRB's

1 Orbiter

1 Integration Cell

#### Launch

Duration - 1

Resources Needed:

1 Orbiter

2 SRB's

1 External Tank

1 Integration Cell

1 Launch Pad

#### Refurbish Orbiter

Duration - 8

Resources Needed:

1 Orbiter

#### Refurbish Pad

Duration - 3

Resources Needed:

1 Launch Pad

### 3.3.3 General Description of the Hypothetical Scheduling System

A job or an activity is described by specifying its name, duration and the resources which must be available for it to be performed. The scheduling and resource allocation system is designed to choose start

times for a set of jobs and assign specific resource items to each job. The system works in two phases. In the first phase (Project Scheduling Timeliner) start times are chosen to minimize the time at which all jobs are completed while satisfying the following constraints:

- a. early start  
(e.g., job A may not start before time = 5)
- b. sequencing or precedence constraints  
(e.g., job B may not start before job A is finished)
- c. resource type availability  
(e.g., is some truck available from a pool of trucks from day 2 through day 4?)

and making a limited attempt to satisfy the constraint:

- d. late start  
(e.g., job A must start at or before time = 10).

In the second phase (Specific Resource Allocator), specific resource items are assigned to each job without moving any of the jobs on the timeline. For example, suppose that job A needs the resource type: truck. Using the Timeliner job A is scheduled when some unspecified truck is available. Specific Resource Allocator assigns a specific truck (truck #72) to job A. It is possible to specify that the same truck used for job A must also be used for job B.

#### Comments

1. Some constraints the user might desire may not be handled by the software of the scheduling system. The user must therefore iterate with the system, changing the real constraints to simplified

constraints until the real desired constraints are satisfied. To aid in this process, one table of suggestions is provided for modeling temporal relations and another for modeling resource relations which cannot be handled directly by the system software. Examples of common modeling difficulties are:

- o Resources with special descriptors are required:  
e.g., job A needs a truck which is filled with fuel (an unfilled truck is not sufficient)
- o Special Timing relations are required:  
e.g., job A must start within 2 days after the start of job B.

2. It is entirely possible that one run of the system will yield a schedule which is partially adequate. In that case, it may be advantageous to freeze part of the jobs on the timeline manually and schedule the remaining jobs using the scheduling system.

The two phases of the system may be executed in one operation or separately as the user desires. If the user desires the Specific Resource Allocator to begin immediately upon completion of the Timeliner, it must be stated explicitly.

The explicit inputs and outputs for each phase of the hypothetical scheduling system are described in the following paragraphs.

## Phase I - Project Scheduling Timeliner

### Inputs:

For each job

- o job name                      - each job must have a unique name

(input those desired from the following list)

- o early start time            - job may not start before this time
- o late start time            - job must start at or before this time
- o start time                 - if the start time is input for a job, it will  
be scheduled at this time regardless of re-  
source or temporal constraint violations. The  
jobs whose start times are specified are placed  
on the timeline first with the other jobs  
scheduled around them.

There are three cases which can be described using the resources for a job.

### Required Resources

If a job requires a resource but returns that resource to the pool of available resources when the job is finished, the resource is called a required resource.

e.g., required resource = 1 truck

If, at the completion of the job, the resource does not return to the availability pool, it is called a deleted resource.

e.g., job A = drive truck to project, required resource = 10 gal. of gas,

deleted resource = 10 gal. of gas.

If, at the completion of the job, a new resource is made available to the pool, it is called a generated resource.

e.g., job A = receive and checkout shipment of trucks, required resources  
= 1 warehouse, generated resources = 3 trucks.

A resource for a given job can be specified as required, deleted or generated.

o available resources - number of units of each resource type available as a function of time.

e.g., trucks = 5 time 0 to time 10

= 3 time 10 to time 36

etc.

#### Outputs:

o job start times (for all jobs)

o violated constraints - the ability to input start times and late start constraints makes it possible that not all constraints will be satisfied. Those constraints which are not satisfied will be enumerated.

o resource profiles - for any resource type, a list may be printed out with the number of units of that resource type used during each time interval. The list must be requested for those resources desired.

## Phase II - Specific Resource Allocator

**Function:** Assign specific resource items from available resource sets to jobs whose start times have already been specified.

### **Inputs:**

For each job

- job name, duration, start time
- job resource requirements  
(i.e., number of each resource types needed plus any descriptors on the required resource)  
(e.g., in "truck, color = blue" the resources type is "truck", the descriptor is "color" and its required value is "blue").

- o available resources - the times that each resource item is available and any descriptors which may be required by a job  
(e.g., Trucks: truck #57, available time 0 → time 50, color = blue, capacity = 1 ton).

### **Output:**

- assignment of a specific resource item to each resource requirement of each job where an item of the right type with the right descriptors is available during the job interval.
- a list of all resource requirements which could not be met.

#### 3.3.4 Results

The results of the paper simulation of the project scheduling problem indicated that the decomposition solution strategy did enable the participants to arrive at successful solutions to the problem. All participants used an iterative approach to their solutions. They agreed that an interactive system in which the computer performed tasks, such as checking resource availabilities and assigning job start times while the human resolved scheduling conflicts was an effective way to solve problems of this type.

Because the decomposition strategy was effective in providing the simulation participants with successful solutions to the project scheduling problem, this strategy was implemented in a batch iterative computer program. In addition, an interactive system was designed based upon this approach and has been successful in solving problems of this type.

3.3.4.1 Decision Tables as an Aid to Iterative Resolution of Modeling Temporal and Resource Constraints - In the project scheduling simulation, decision tables were developed for the participants to aid them in determining an appropriate next iteration. In every case, the subject elected not to use the tables to arrive at their solutions; however, it was found that the decision tables did accurately illustrate the solution process that they chose to employ. These tables are included, therefore, as documentation of appropriate logical sequences used to reach an iterative solution to a project scheduling problem.



To explain how these decision tables are interpreted, a nonsensical example is presented below:

Get Up or Go Back To Sleep?				
Conditions:	Is it either Saturday or Sunday?	N	N	Y
	Is it later than 6:30 a.m.?	N	Y	-
<hr/>				
Suggestions:	Get up, go to work	N	Y	N
	Go back to sleep	N	N	Y

The table is used as follows. The conditions are questions which the user can answer to determine which suggestions are applicable to his situation. The questions should be answered sequentially from top to bottom, since the answer to one question may make another question below it irrelevant.

The columns of the table are searched until a column which accurately reflects the constraint situation is found. The suggested actions are then found below the double line in the same column.

In the example, the table suggests only whether to "get up" or "go back to sleep." The first question is "Is it either Saturday or Sunday?" If the answer is no, then the next question should be answered, "Is it later than 6:30 a.m.?" If both questions are answered no, the suggestion is to return to sleep. If the second answer is yes, the suggestion is to get up.

If the answer to the first question is yes, (it is Saturday or Sunday), the second question is ignored. No matter what time it is,

go back to sleep. The dash (-) symbol is used in column three to show when a question is to be ignored.

The decision table provides a column of suggestions for every combination of answers to the condition questions. If there were no dashes, there would be  $2^N$  columns, where N is the number of condition questions. Since the dashes represent both a yes and no answer to a question, one column may serve for more than one answer combination.

3.3.4.2 Application of Decision Tables to the Project Scheduling Problem - The following decision tables list common difficulties encountered when trying to model the temporal or resource relations of a problem to input to the Project Scheduling Timeliner. The difficulties may become evident at the first modeling or after several iterations. In either case, suggestions are provided which should help in resolving the modeling problem. The suggestions are related to the difficulties through decision tables with expanded descriptions of both suggestions and difficulties following.



# TEMPORAL RELATIONS: DECISION TABLE

S = Start time  
Y = Yes  
N = No  
- = No or Yes, Skip Question

## Difficulty Conditions

1. Are you having trouble modeling the constraint $S_X \leq S_Y + C$ ?	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
2. Are you having trouble modeling the constraint $S_X \leq S_Y + C$ ?	N	N	N	N	N	N	Y	Y	Y	Y	N	N
3. Are you having trouble modeling the constraint $S_X = S_Y + C$ ?	N	N	N	N	N	N	N	N	N	N	Y	Y
4. For the constraint you are trying to model, is $C \leq d_Y$ ?	N	N	N	Y	Y	Y	N	N	N	Y	-	-
5. For the constraint you are trying to model, is it necessary for job X and job Y to be on-going concurrently?	N	Y	Y	N	Y	Y	N	Y	Y	-	-	-
6. For the constraint you are trying to model, it is unacceptable for the predecessors to job X to be forced to be completed before the start of job Y?	-	N	Y	-	N	Y	-	N	Y	-	N	Y

Suggestions (See expanded explanation in next section)

A. Make job Y a predecessor of job X				Y								
B. Make job X a predecessor of job Y	Y											
C. Give jobs X and Y common predecessor and successor sets	Y		Y									
D. 											Y	
E. 							Y					
F. Combine jobs X and Y into one job	Y	Y		Y	Y		Y	Y		Y	Y	
G. Combine predecessors of job X into the job X-job Y super job		Y			Y			Y				Y

\* Blank boxes under suggestions are equivalent to no'

EXPANDED EXPLANATIONS FOR  
TEMPORAL RELATIONS TABLE

- D. Create a dummy job which uses no resources but has the proper duration. Make job Y a predecessor of the dummy job. Make the dummy job a predecessor of job X.
- E. Create a dummy event (a job using no resources and of zero duration) and a dummy job. Make the dummy event a predecessor of the dummy job and job Y. Make the dummy job a predecessor of job X.
- F. Combine jobs X and Y into one job. Make all predecessors of either job predecessors of the combined job. Specify the relationship between the two job start times.
- G. Combine predecessors of job X into the conglomerate of job X - job Y, setting the temporal relationship between the start times of each job.

# RESOURCE RELATIONS: DECISION TABLE\*

1. Explicit descriptors satisfied?	N	N	N	N	N	N	Y	Y
2. Does the resource relation you're trying to model involve a descriptor which has more than two values?	N	N	N	Y	Y	Y	-	-
3. Are the commonality constraints satisfied?	N	N	Y	N	N	Y	N	N
4. Is the resource which is to be common to several jobs to be available during the times those jobs are not being accomplished?	N	Y	-	N	Y	-	N	Y
<hr/>								
A. Replace resource descriptors with appropriate precedence constraints.	Y	Y	Y	N	N	N	N	N
B. Assign specific resources by input specification.	N	Y	N	Y	Y	Y	N	Y
C. Use deleted and generated resources.	Y	N	N	Y	N	N	Y	N

\* See expanded explanations in following section.

EXPANDED EXPLANATIONS FOR  
RESOURCE RELATIONS TABLE

1. & 2. Explicit Descriptors

1. Are explicit descriptors satisfied?

It is possible to specify that the resources needed for any job have certain characteristics; for example, job A needs a truck that is one ton in capacity and red. While the specific resource allocation of phase 2 attempts to find a truck meeting all the characteristics, the timing of jobs is done considering only that job A needs a truck. The statement "explicit descriptors satisfied" refers to the fact that job A may have been scheduled (timelined) successfully, but when the specific resource allocator sought a truck meeting all the requirements, none could be found.

2. Does the resource relation you're trying to model involve a descriptor which has more than two values?

e.g., the resource truck may have the descriptors fueled or unfueled and is always in one of these two conditions.

3. & 4. Commonality

3. Are the commonality constraints satisfied?

It is often necessary to require that two jobs use not only the same resource type, but the same resource item (e.g., jobs A and B both need a truck, in fact the same truck). It is possible to specify that jobs require the same item of any resource type.

However, the Timeliner only takes into account the requirement for the same resource type. The specific resource allocation of phase 2 does seek resources which meet the common-item requirements, but it may not be possible to meet them all.

4. Is the resource which is to be common to several jobs to be available during the times those jobs are not being accomplished?

There are two conditions which usually cause the need for a commonality requirement on a resource.

Condition 1 (Example)

Job A is "Deliver power generator to construction team"

Job B is "Pick up power generator from construction team"

The resource is truck driver and it is desired to use the same truck driver for both jobs, since he knows exactly where he left the generator. However, the truck driver is free to do other jobs in between jobs A and B.

Condition 2 (Example)

Job A is "load the truck"

Job B is "drive the loaded truck to its destination"

Job C is "unload the truck"

The same truck is desired for Jobs A, B and C. Additionally, once the truck begins loading (job A), it is unavailable for any other jobs until it is unloaded (job C).

- A. Replace resource descriptors with appropriate precedence constraints. Example:

Rather than using the modeling -

job A, required resource = truck, condition of truck = filled  
with fuel;

Use the modeling -

job A, required resource = truck, predecessor = job B.

job B = fill truck with fuel, required resource = truck.

- B. Assign specific resources by input specification. Example:

job A, required resource = truck #57

job B, required resource = truck #57

Specific resource allocation will fill the specific resource demands  
before allocating specific resources to general resource demands.

- C. Use deleted and generated resources.

Example:

job A = load the truck

job B = drive the truck to its destination

job C = unload the truck

Specify

job A, required resource = truck. Deleted resource = truck.

job B, (no required resources)

job C, (no required resources), resources generated = truck.

This will cause a truck to become unavailable to other jobs at  
the start of job A and become available again at the completion  
of job C.



### 3.4 A MAN-COMPUTER STRATEGY FOR SOLVING A LARGE CLASS OF SCHEDULING PROBLEMS

The results of the Project Scheduling paper simulation which allowed the user only two simplified capabilities suggested the development of a complete problem solving strategy. The ability to develop a classification for unsatisfied constraints and to define an appropriate response (which did correlate with user preference) for each combination of constraint violations suggested that the strategy was reasonably robust. A more complete description of this strategy is now presented. The following material has been published by O'Doherty<sup>1</sup>.

#### 3.4.1 Concept

Producing large schedules and resource utilization assignments in a sufficiently short time will require the aid of a computer. One possibility considered for computerized scheduling was a heuristic procedure which would try to assign specific resources to an activity at the time it was placed on the timeline. However, it was recognized that several attempts may be required before a time which satisfies the constraints is found. Thus a number of unnecessary resource assignments would be made during the timelining of each activity. Since the bookkeeping involved in assigning specific resources to each item is considerable, it was decided that a reasonably good timeline should be developed before an attempt is made to assign specific resource items to the activities on that timeline. The

---

<sup>1</sup> R. J. O'Doherty, "The Use of Project Scheduling for More General Problems: An Application to the Space Transportation System" ORSA/TIMS National meeting, Chicago, Illinois, 30 April 1975.

potential of this logical decomposition was suggested by the paper simulation described in Section 3.3. A more detailed description of an implemented system based on this strategy is now presented.

#### 3.4.2 Functional Design

Classically, project scheduling heuristics have dealt with the simplified problem of scheduling a network of activities which have durations and require resources of different types. The network defines the sequence in which the activities must occur through the use of predecessors. For example, in the network of Figure 3-2, activity A must be completed before activities B or C may start and both activities B and C must be completed before activity D may start (A is a predecessor of B, A is a predecessor of C, B and C are predecessors of D).

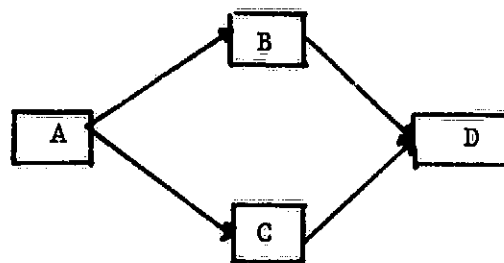


Figure 3-2 Sample Network

In addition to the network or sequencing constraints on the allowable start times for activities, resource constrained project scheduling deals with the fact that only limited resources are available for the project and these must be shared by the various project activities. Project scheduling resource constraints are of the pool type. That is, the activities are assumed to require resources

which may be drawn from resource pools (e.g., trucks). As long as a sufficient number of units of the required resource type is available in the pool during any time period, it is assumed that the resource requirements of the activity can be met during that time period. The constraint is on the size of the pool or the total number of units of each resource type which are available to all activities in the project during any one time period.

Since precedence constraints are the only temporal constraints and pool level limits are the only resource related constraints considered in the project scheduling model, the philosophies of the time-progressive heuristics can be used to produce computationally efficient algorithms which satisfy all the constraints. However, these are not the only constraints which face the flight scheduler or most industrial schedulers. For example, launching a flight within a launch window presents the problem of an activity with rigid constraints on the earliest and latest time it may start. The requirement to retrieve solid rocket boosters from the ocean after the launch and before they sink, presents the problem of scheduling one activity within a limited time after the occurrence of another activity. In general, Temporal constraints fall into two categories: Those which say that an activity may not start before a certain time, and those which say an activity must start before a certain time. Those of the first type include the opening of a launch window and can usually be handled using dummy activities (activities which require no resources) as predecessors to real activities. Those of the second type, including the closing of

the launch window and recovering the solid rocket boosters, directly conflict with the philosophies of project scheduling heuristics. While the heuristics try to schedule activities "as soon as possible", the activities still may be delayed indefinitely without violating any precedence constraints. It is possible to handle some of the "must start" constraints through modeling (see the next section) or by increasing the complexity of the heuristic algorithm (see Burman<sup>1</sup>). However, it is not possible to handle all "must start" constraints without a significant increase in algorithm complexity which must account for cases in which no solution exists (e.g., an activity must be started before time = 5; however, it has a predecessor whose duration is 10).

Just as precedence constraints are not the only temporal constraints facing the flight preparations scheduler, pooled resources do not describe all his resource requirements. Usually in a project scheduling heuristic, the pooled resource required by an activity is returned to the pool as soon as the activity is completed (e.g., trucks in a truck pool) or the resource is never returned at all (e.g., gasoline). The situation for resources which are used over and over again but must be refurbished after each use is between these two extremes. For example, launch pads and indeed most the critical resources involved in scheduling flight preparations are reuseable but require refurbishment. If the launch pad is returned

---

<sup>1</sup> Burman, P. J., "Precedence Networks for Project Planning and Control," McGraw-Hill Book Company (UK) Limited, 1972.

to the pool immediately, another launch may be scheduled before the launch pad can be made ready. Fortunately, Project Scheduling Timeliner allows activities to "create" resources, a provision common in commercial programs. Using this capability, the refurbishment situation can be modeled within the project scheduling framework.

Other difficulties arise when trying to assign specific resource items to activities which have been timed using a project scheduling heuristic. The Specific Resource Allocator assigns resources to activities as they occur on the timeline. Once an activity is chosen, the Allocator searches through the available resources until one is found which is available for the duration of the activity. However, there may be a requirement to use the same resource item for several jobs. For example, suppose there are two activities: "prepare the rocket for launch" and "launch the rocket". The two activities require a rocket and in fact the same rocket. In this case the Allocator searches for a resource item which is available to all the activities which have been specified as requiring the same item. If this requirement is not taken into account when the timeline is generated, it may not be possible to find a resource item available to all the specified activities. As will be shown in the next section, using the correct modeling procedures, it is usually possible to produce schedules for which resource assignments satisfying all requirements can be made. However, just as some temporal constraints may be violated by timelines produced by the Timeliner, it may not be possible to make assignments satisfying all resource requirements using the Allocator. Thus the system must be used iteratively to produce totally acceptable schedules.

The need for iteration and human intervention to produce final schedules is not a surprising result. Indeed, human schedulers have always adjusted automatically generated schedules to achieve such hard-to-express goals as "don't schedule too much work on the afternoon before a holiday". The automatically generated schedule should be considered a tentative schedule, subject to human evaluation.

The idea of solving problems by generating a good first-cut solution and then reworking smaller parts of the problem to obtain better solutions has been used successfully to partition logic circuits into integrated circuit chips (Hanan, et al<sup>1</sup>) and as a strategy for solving job shop or flow shop problems (Ferguson and Jones<sup>2</sup>; Jones et al.<sup>3</sup>; Dannenbring<sup>4</sup>). Jones et al (1970) discuss an experiment in which a number of subjects were given an interactive computer terminal and asked to schedule the daily activities of a simulated machine shop. Different tentative schedules can be generated by selecting different heuristic rules. The subjects attacked the problem by generating several trial schedules using combinations of the heuristic rules, choosing a "best" schedule from among the trial solutions, and then

---

<sup>1</sup> Hanan, M., Mennone, A., Wolff, P. K. Sr., "Iterative-Interactive Technique for Logic Partitioning," IBM Journal of Research and Development, Vol 18, No 4, July 1974, pp 328-337.

<sup>2</sup> Ferguson, R. R., and Jones C. H., "A Computer Aided Decision System," Management Science, Vol 15, No 10, June 1969, pp B550-B561.

<sup>3</sup> Jones, C. H., Hughes, J. L., and Engvold, K. J., "A Comparative Study of Management Decision-Making from Computer-Terminals," AFIPS Spring Joint Computer Conference, 1970, pp 599-607.

<sup>4</sup> Dannenbring, D. G., "An Evaluation of Flow-Shop Sequencing Heuristics," ORSA/TIMS National Meeting, Boston, 1974.

using the interactive capability of the terminal to fine-tune the "best" schedule. Currently we are in the process of adding allowable priority indices to the Project Scheduling Timeliner and giving the system an interactive graphics capability so that a similar approach may be used for scheduling flight preparations and projects in general.

The philosophy for using the system then is as follows. First of all, model as much of the problem as possible within the project scheduling framework. Secondly, generate a tentative schedule using the Project Scheduling Timeliner. If the temporal constraints of the problem are not satisfied by the schedule, several options are open. The activity start times which are satisfactory could be frozen at the values suggested by the Timeliner, additional constraints added to the modeling and the Timeliner used to schedule the remaining activities. The start times of activities involved in difficult constraints could be frozen at values which would satisfy the constraints and the Timeliner used on the remaining activities; or the start times of activities in trouble spots could be shifted manually. Once the timeline satisfies the human scheduler, the Specific Resource Allocator is used to assign resource items to the activities. Here again, if the allocations are not satisfactory, the human scheduler has the option to freeze some of the assignments, manipulate some of the resource requirements and use the Allocator again, or manually manipulate the assignments. The next two sections present a more detailed discussion of the Timeliner and a concrete example of the application of this philosophy to a flight preparations scheduling problem.

### 3.4.3 Implementation

The computer implementation of the decomposition strategy just described has taken the form of three segments of software. The second and third segments correspond to the timeliner and explicit resource allocator alluded to in previous sections. The first segment is an input processor that automatically creates jobs to be scheduled from the input problem specification. This three segment architecture is shown in Figure 3-3. A more detailed discussion of the software segments follows.

The problem chosen to demonstrate the solution strategy is a complex flight assignment problem with resource allocation, designed to be easily modified to make tradeoff analyses and as requirements grow and change. There are two basic flight sequences of operations defined and these can be coupled and modified in any way the user might want to describe his problem. As was mentioned, the code was divided into three parts. The first part is essentially a preprocessor of standard data. The flow chart in Figure 3-4 illustrates the logic flow. The first step is to copy into local storage only that part of the data base necessary to solve the current problem. From the resultant data subset, a jobset is generated which contains all operational and resource requirements necessary to schedule each job. All flights are examined to determine whether the jobs in the particular flight sequence are defined correctly for the payloads assigned to that flight. If modifications to the job definitions or resource requirements are necessary, they are made in the jobset. Also, if



### EXPERIMENTAL SCHEDULING SYSTEM

INPUT = Parameters + Resources  
Non-standard Data

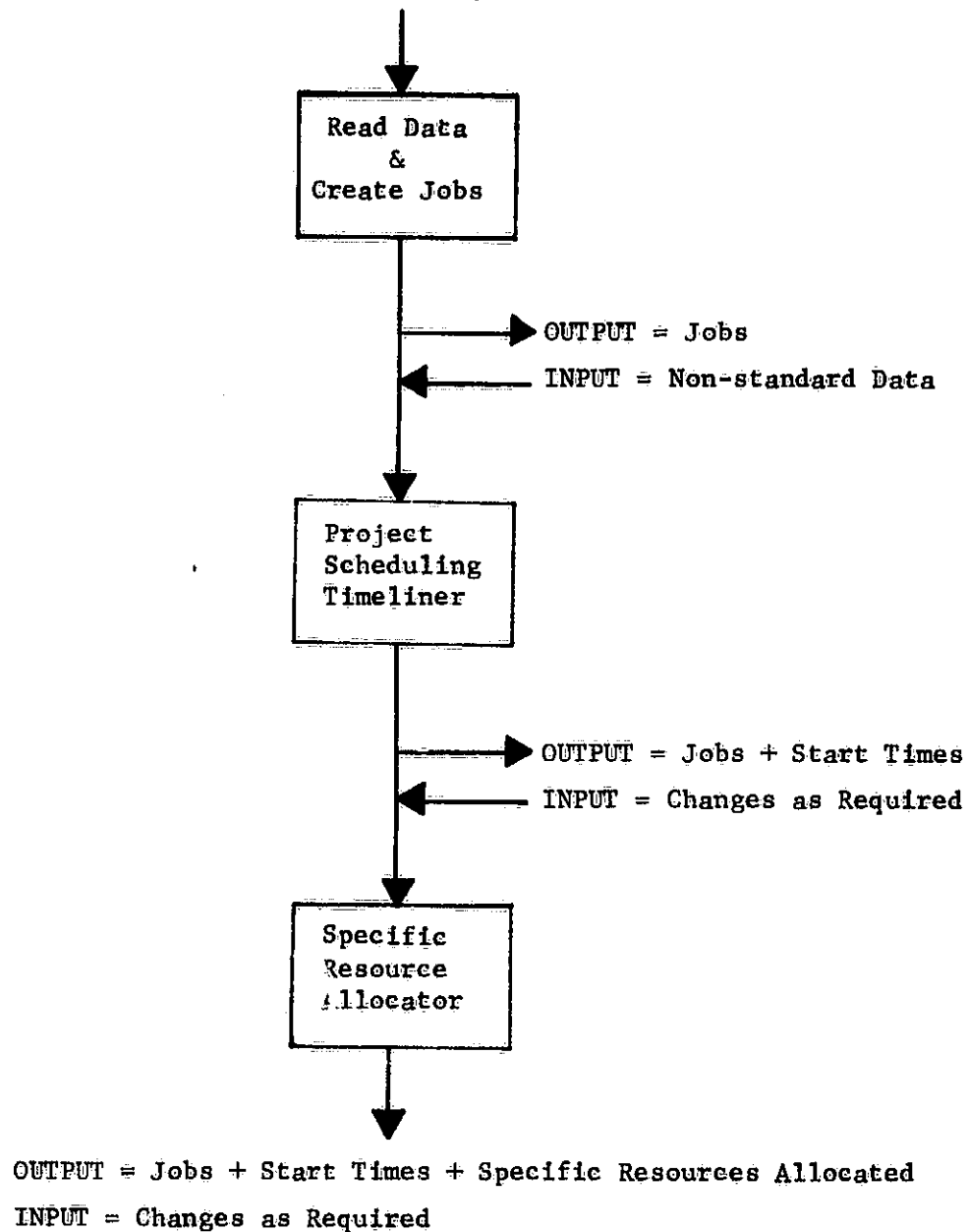


Figure 3-3 Architecture of the Computer Implementation  
of the Decomposition Strategy

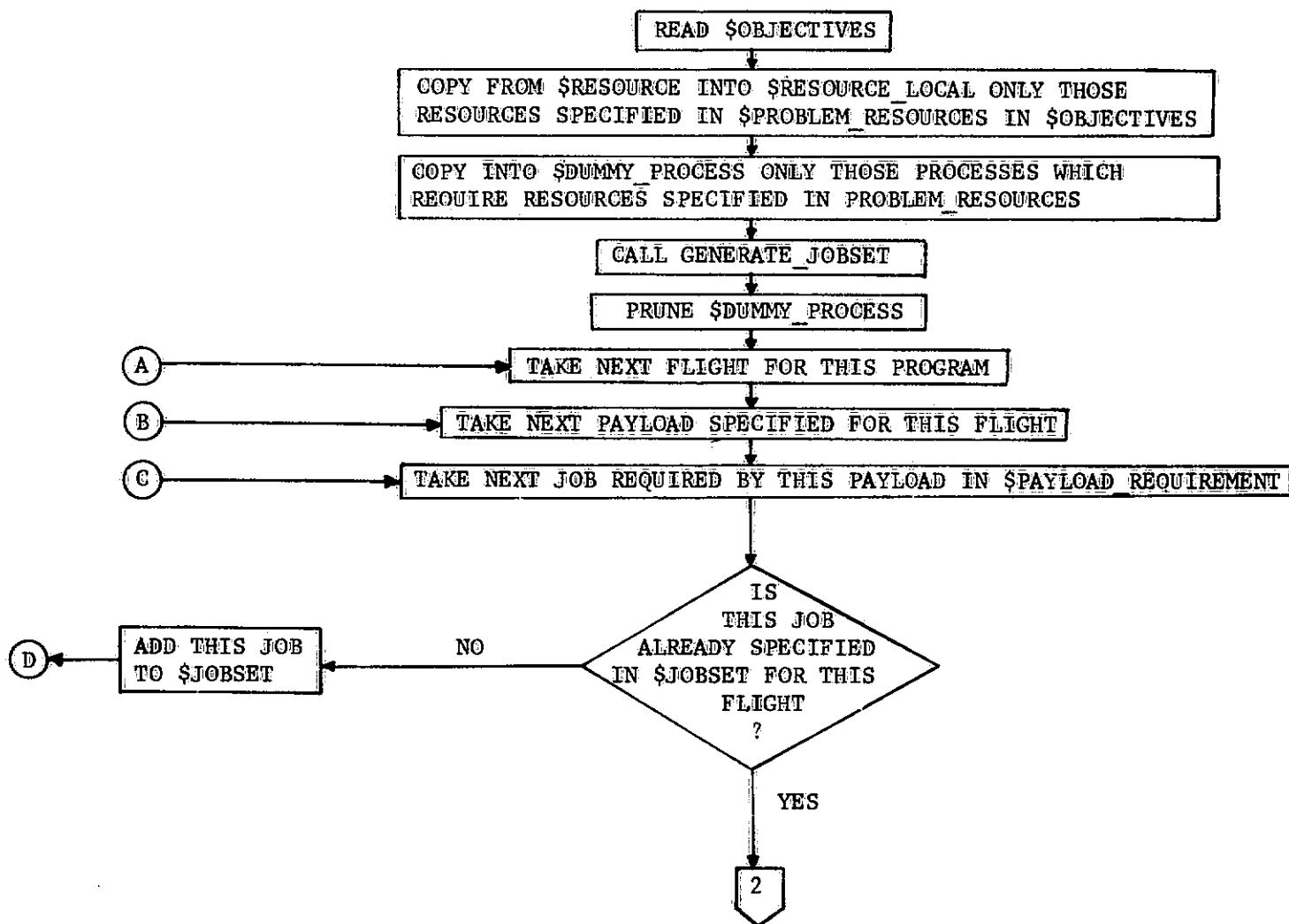


Figure 3-4 Demonstration Program Input Processor Logic

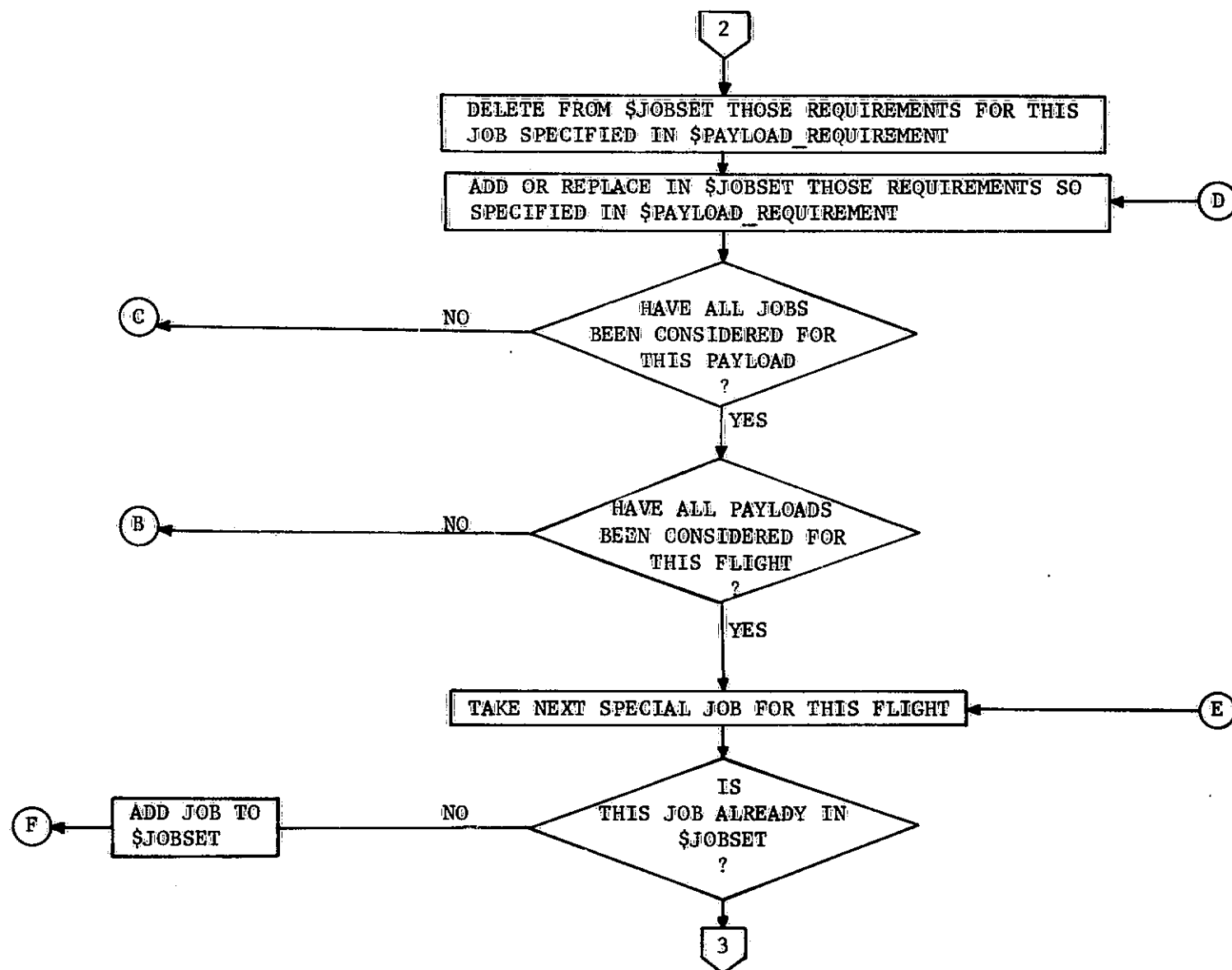


Figure 3-4 (Continued)

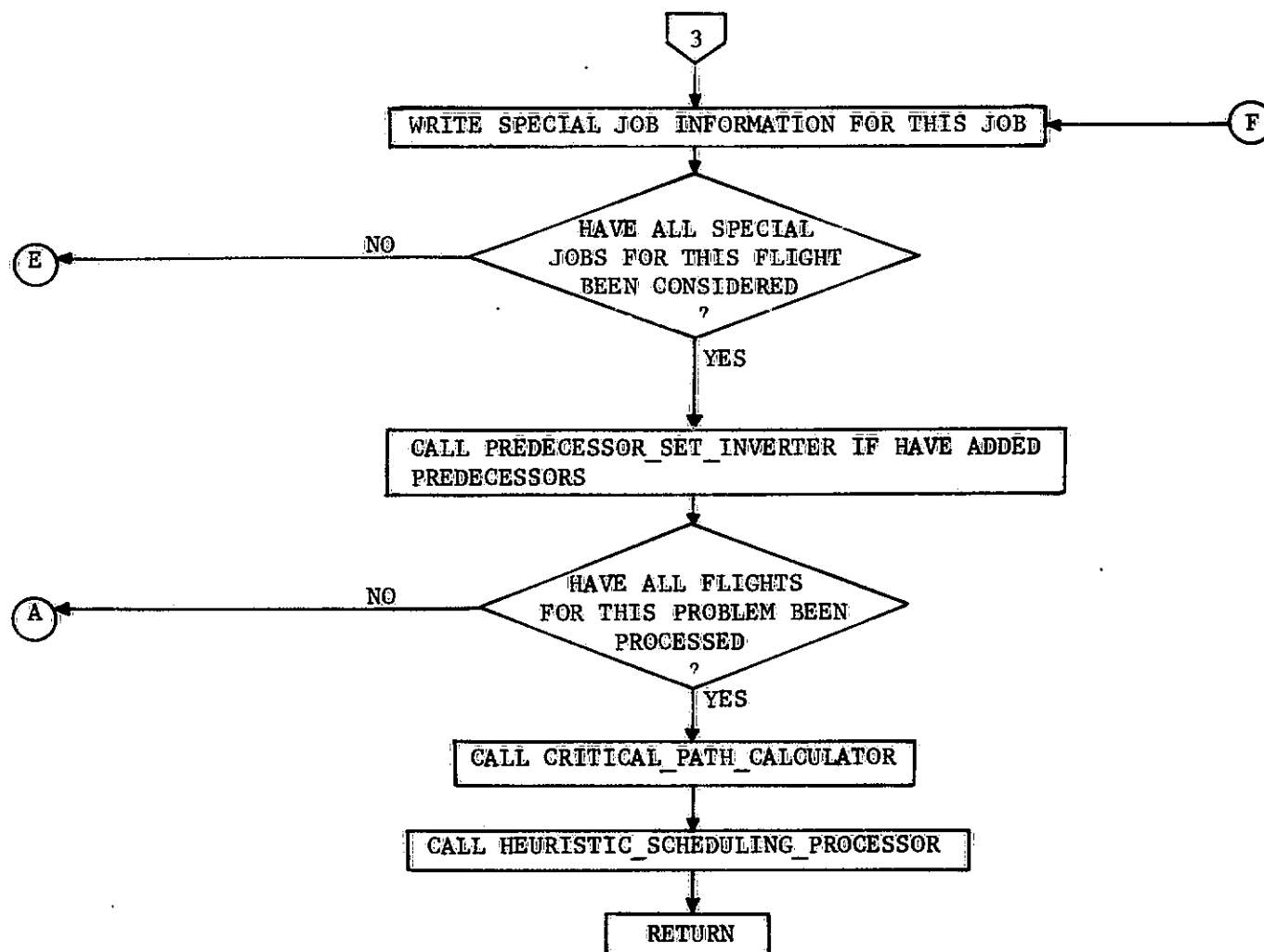


Figure 3-4 (Conclusion)

special jobs are required for the flight, they are added into the jobset.

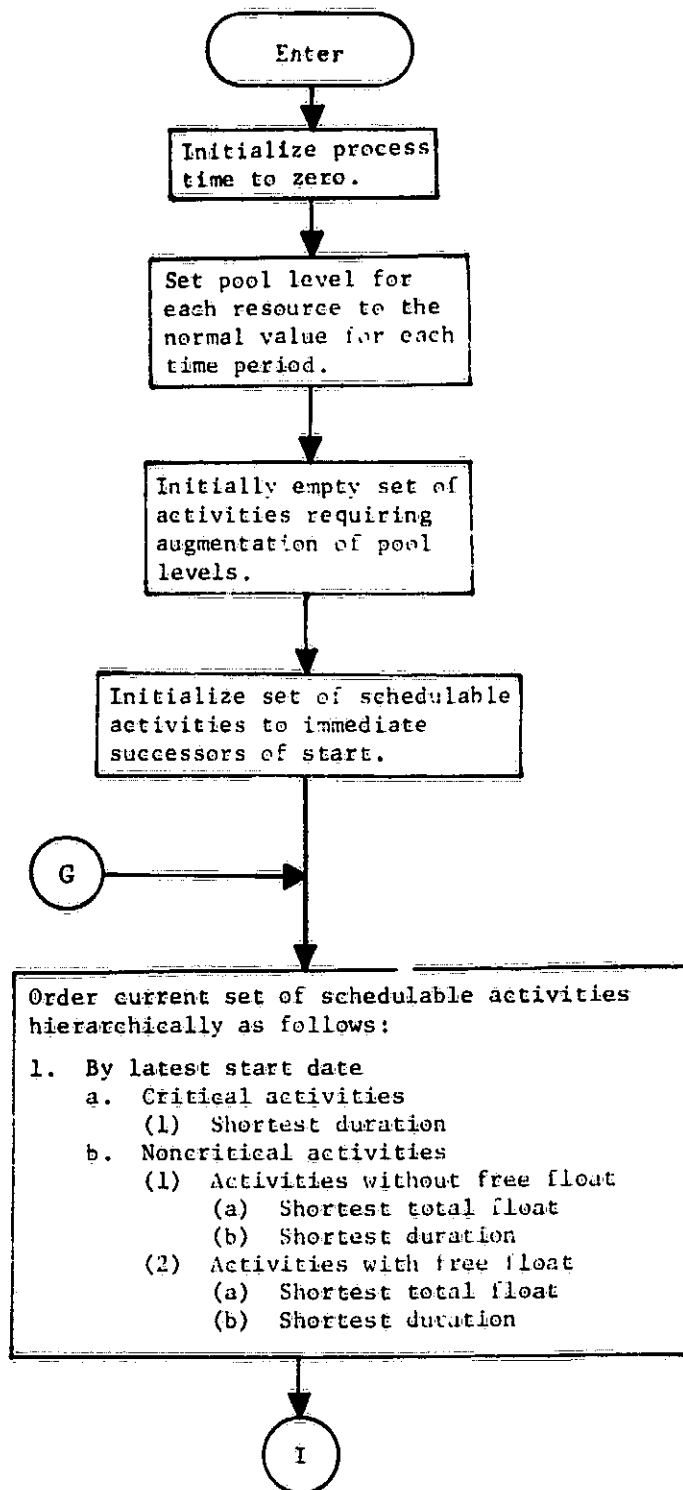
The second part is the timeliner where the schedule is built based on pooled resource availability. The logic flow is illustrated in Figure 3-5.

The Project Scheduling Timeliner basically works by stepping through time, scheduling eligible activities from a prioritized list. The activities are considered eligible as soon as time reaches the early start limit for the activity. The activities are prioritized based on late start time limit, slack and activity duration. Early start time limit, late start time limit and slack are initially determined by the Critical Path Method (CPM; see Kelley<sup>1</sup>).

CPM determines an early start limit and late start limit based solely on the activity durations and the network or predecessor constraints. The early start limit is determined as the earliest time all the predecessors of an activity could be finished, the latest of the predecessor early finish times. The latest of the early finish times for all the activities is the earliest possible finish time for the project, the time at which the project could be finished if there were no resource constraints. Once the earliest project finish time has been established, CPM works backwards through the project activities finding the latest time at which a job could be started without

---

<sup>1</sup> Kelley, J. E., Jr., "The Critical-Path Method: Resources Planning and Scheduling," Chapter 21 of Industrial Scheduling, J. F. Muth and G. L. Thompson, ed., Prentice-Hall, Englewood Cliffs, New Jersey, 1963.



ORIGINAL PAGE IS  
OF POOR QUALITY

Figure 3-5 Scheduling Processor Logic

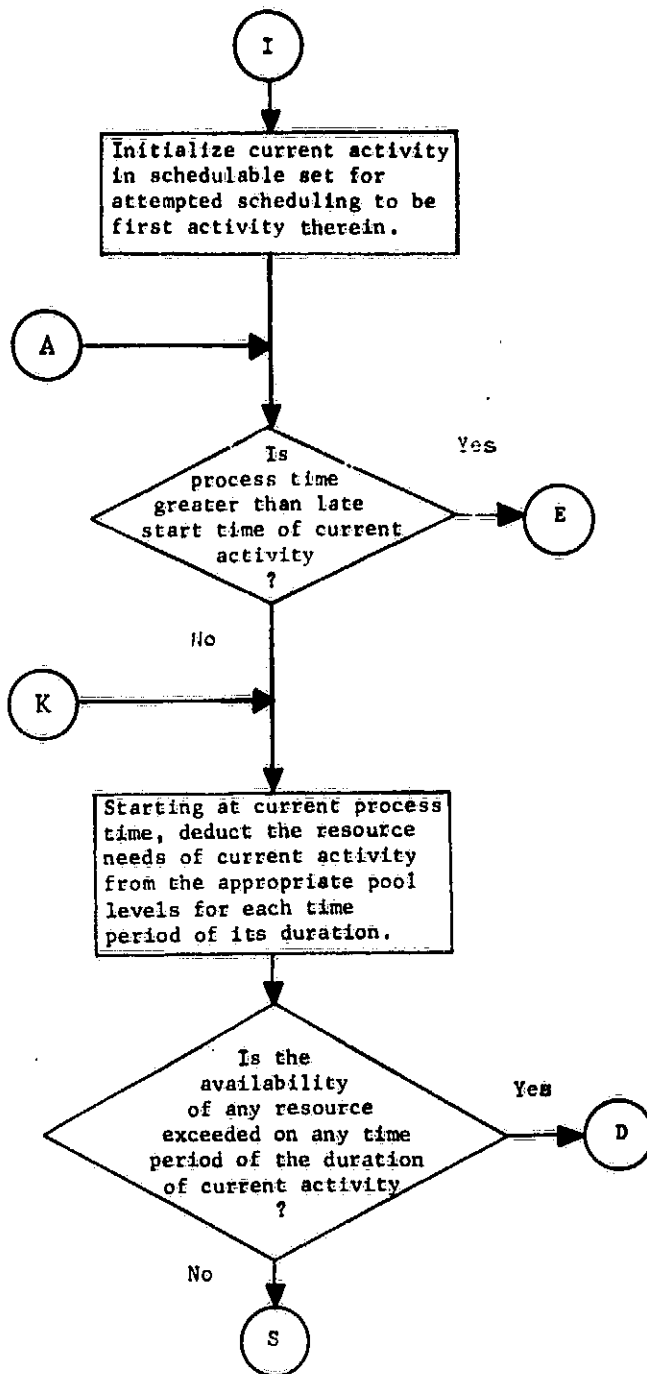
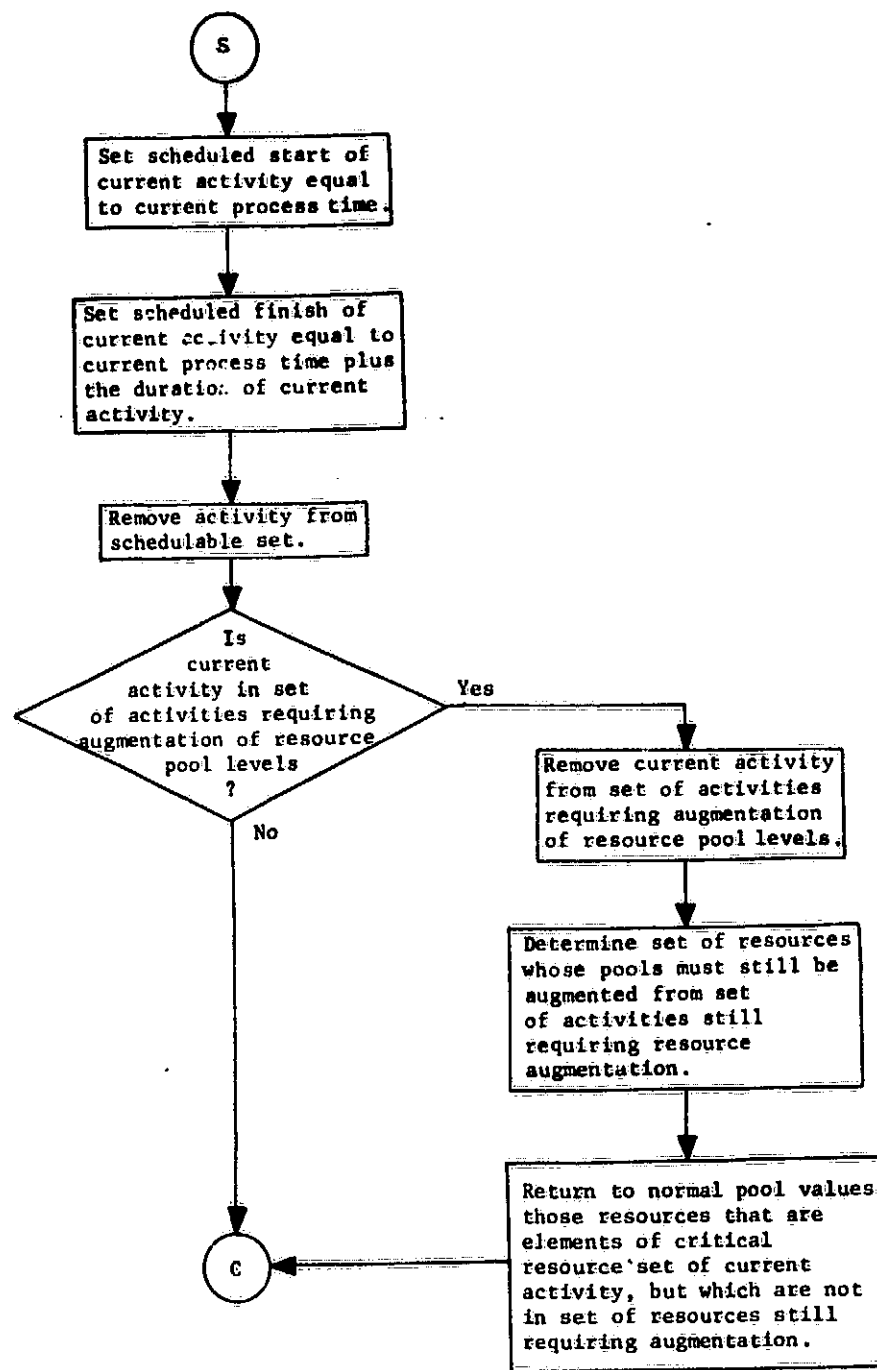


Figure 3-5 (Continued)



ORIGINAL PAGE IS  
OF POOR QUALITY

Figure 3-5 (Continued)



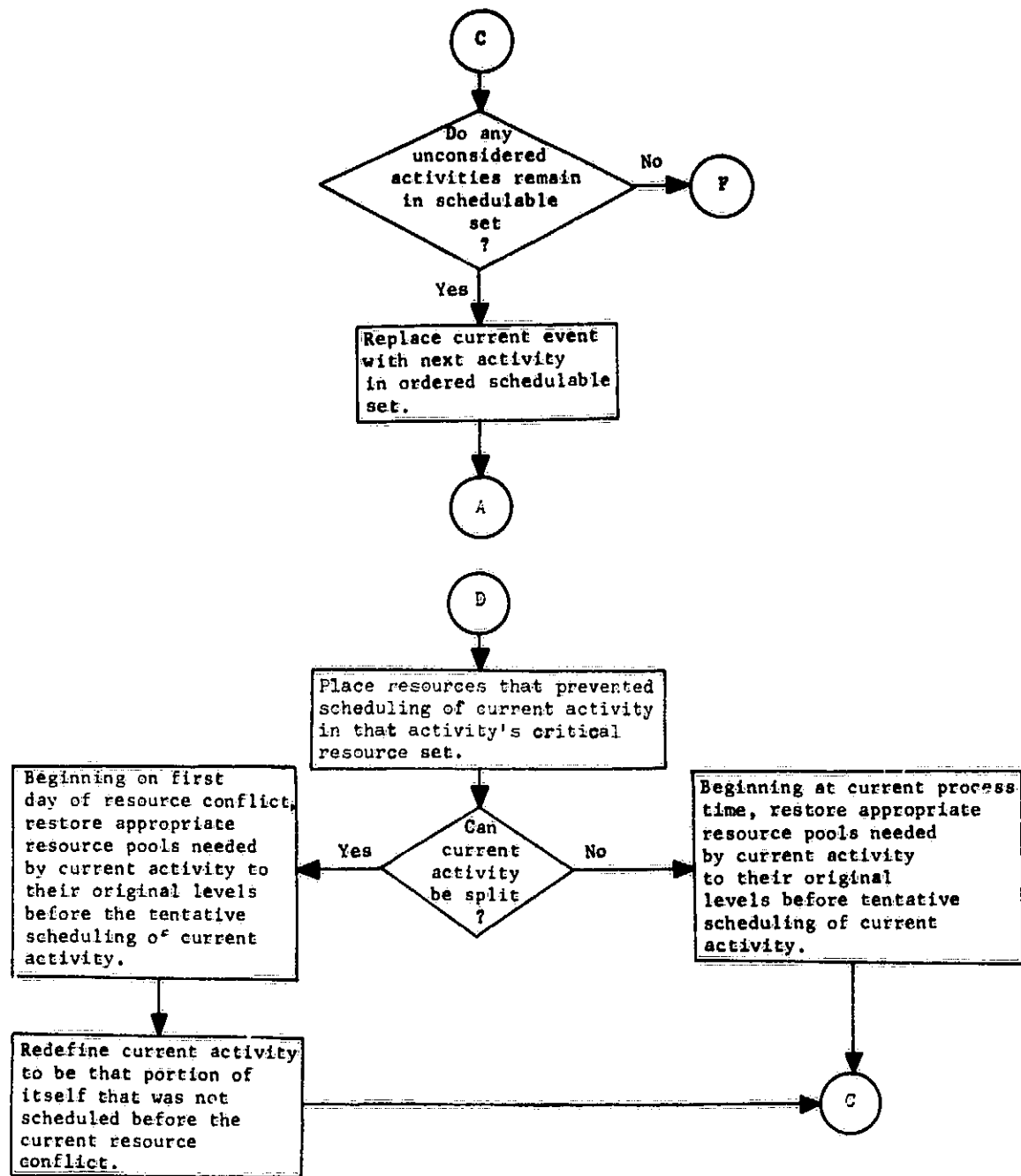


Figure 3-5 (Continued)

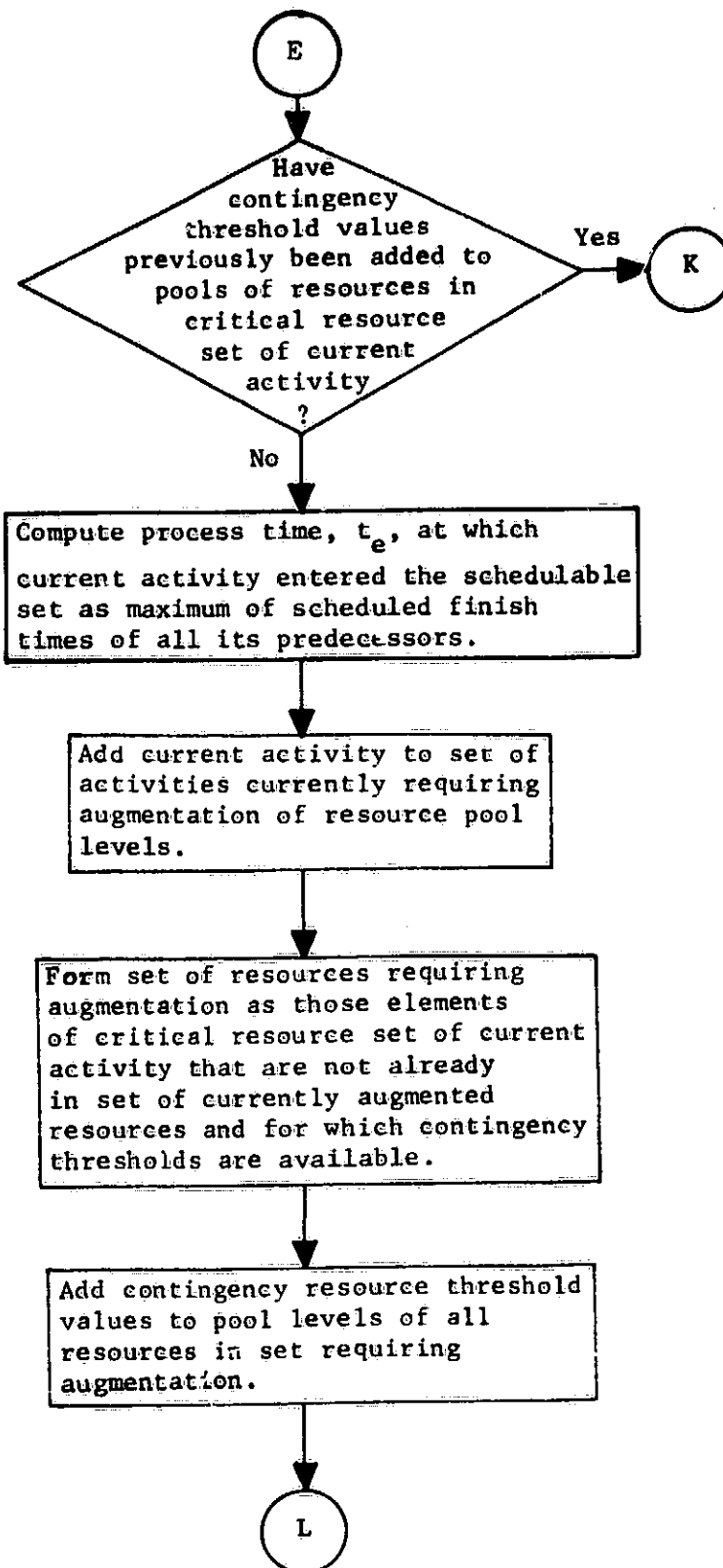
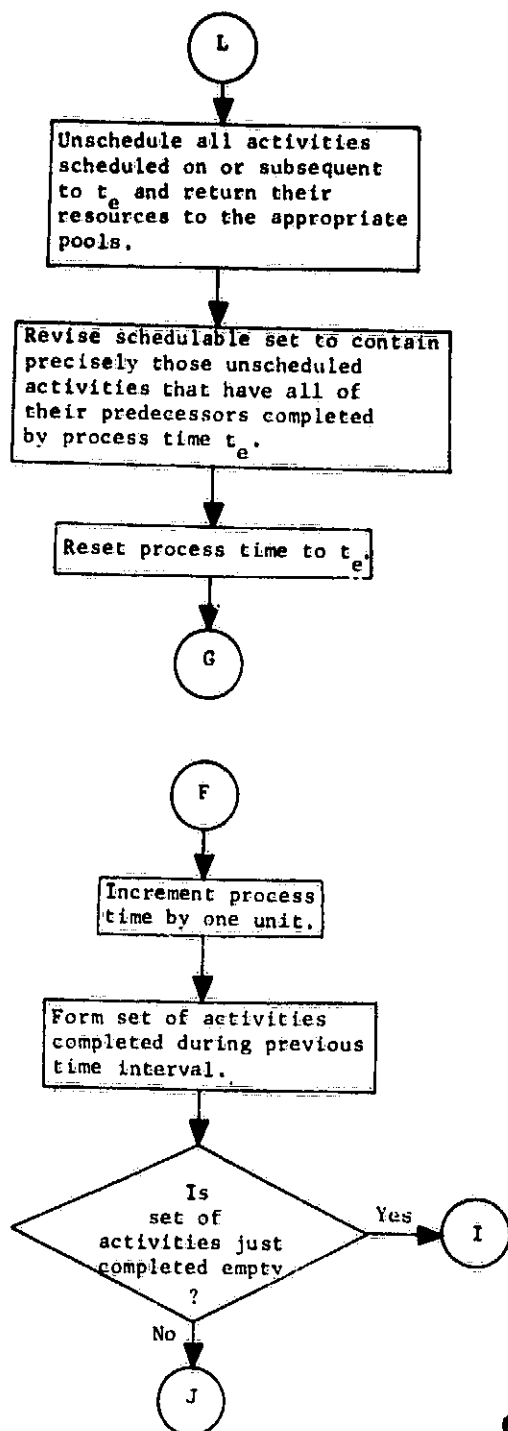


Figure 3-5 (Continued)



ORIGINAL PAGE IS  
OF POOR QUALITY

Figure 3-5 (Continued)

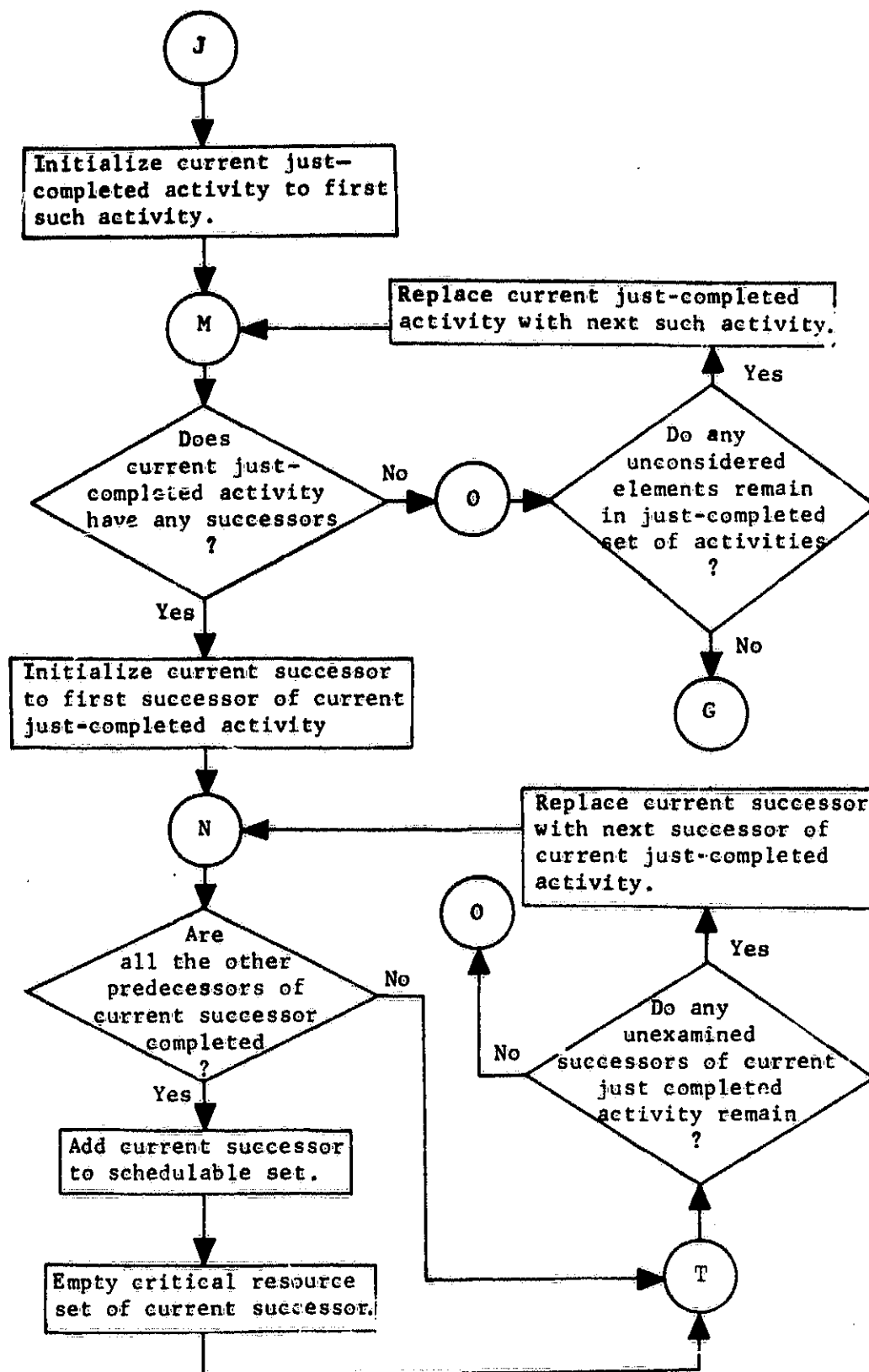


Figure 3-5 (Conclusion)

increasing the project early finish time. This is determined recursively as the latest time an activity could be started so that none of its successors (if A is a predecessor of B, B is a successor of A) are forced to delay the project early finish time. The difference between the early and late start limits is the slack of the activity or the size of the time interval within which the activity can be scheduled without impacting the project early finish. Slack is a measure of the stringency of the temporal constraints on an activity. Those activities whose slacks are zero must be scheduled as soon as their predecessors have been finished if the project early finish time is to be maintained and are said to be on the critical path.

If, in addition to the predecessor constraints on an activity, there were also a direct constraint on the early start time, it could be modeled by using a dummy predecessor. The duration of the dummy activity would be the difference between the early start time limit and the start time for the schedule. Since the constrained activity cannot start until all its predecessors are complete, the early start limit which results from CPM will be at least as great as the direct early start limit. The same effect can be achieved by allowing the user to input the early start time limit directly to the Project Scheduling Timeliner and making a minor modification to the CPM calculations. After the latest of the early finish times of all predecessors of an activity has been found, it is compared to the direct early start limit and the more stringent or later of the two limits is used as the early start limit.

Direct constraints on late start limits could also conceivably be modeled using dummy activities. The duration of the dummy would be the difference between the project early finish time and the late start limit; but the project early finish time is not known a priori. However, an approach similar to that used for direct constraints on early start limits can also be used for late start limits. The earliest of the late start limits of the successors of an activity is compared to the direct late start limit and the more stringent or the earlier of the two limits is used. Now if an activity is started at a time later than its late start limit, it means that either the project early finish time will be increased or that an input late start limit will be violated. Slack retains its value as a measure of the stringency of the temporal constraints (network, early start and late start).

Direct input of late start limits makes it possible that an activity's slack may not only be zero but negative (e.g., an activity has a late start limit of 5 and a predecessor of duration 10. The slack is -5). In such cases it is not possible to satisfy all the temporal constraints even with unlimited resources. Timeliner allows the user to choose whether to terminate execution if negative slack is found or continue.

Once the early start, late start and slack have been determined for all activities, the Timeliner begins to step through time, scheduling activities from a prioritized list at each time point. Activities from the list are scheduled at the current time point as long as resources are available. If an activity is not scheduled at its

early start time, its slippage may affect the early starts and thus the slacks of its successors. If the activity has negative or zero slack and must be slipped, the project length may be extended and the late starts and slacks of other activities may require adjustment. Timeliner dynamically adjusts early starts, late starts and slacks to account for slippage beyond the early start limits of activities. However, it does allow activities to be slipped beyond their original late start limits.

To determine whether sufficient resources are available to schedule an activity at a given time, it is necessary to know what is needed and what is available. The amount of each available resource is specified as a pool level and is allowed to vary from time unit to time unit. Timeliner also allows the user to specify for each resource needed for an activity the time interval or intervals over which the resource is needed, and an initial and final quantity for each time interval. The initial quantity is the amount which will be subtracted from the resource pool for the duration of the interval. The final quantity is the amount which is returned to the pool at the end of the interval. If no interval is specified, it is assumed that the resource is required for the duration of the activity. As will be shown in the example, the capability is very useful in modeling complex activities and refurbishment.

The third part of the batch flight support program explicitly allocates the resources to the jobs on the schedule built by the timeliner. The first step in this process is to build the resource

commonality groups (see Figure 3-6). All jobs for each flight requiring resources of the same type are grouped together to insure that they will all be allocated common resources. Once these requirement groups are generated, then the explicit allocation of resources begins (see Figure 3-7).

The schedule is examined taking each job in order. Each resource type required by this job is examined. A determination is made whether or not this job belongs to a commonality group for this resource type. If it does not and the user has explicitly allocated a resource to fill this need, that resource is checked to see if it is available and in the proper state for assignment to this job. If it is not, contingency levels are checked. Finally, if the resource can be assigned to this job, the necessary bookkeeping is done to record the fact that this resource is assigned for the duration of the job's requirement. If this resource is not available or not in the proper state for assignment to this job, that fact is output. Whatever the result, the algorithm then proceeds to the next resource type for this job.

If the user has not specified a resource, and the job is still not a member of a requirement group, the available specific resources are examined to see if the job's requirements are met by any unassigned resources. If there is an available resource satisfying the job's requirements, its name or identification is recorded and the necessary bookkeeping is done. If there are no available resources, then contingency levels are examined. If there still are no resources of this resource type available for this job, that fact is output. In either



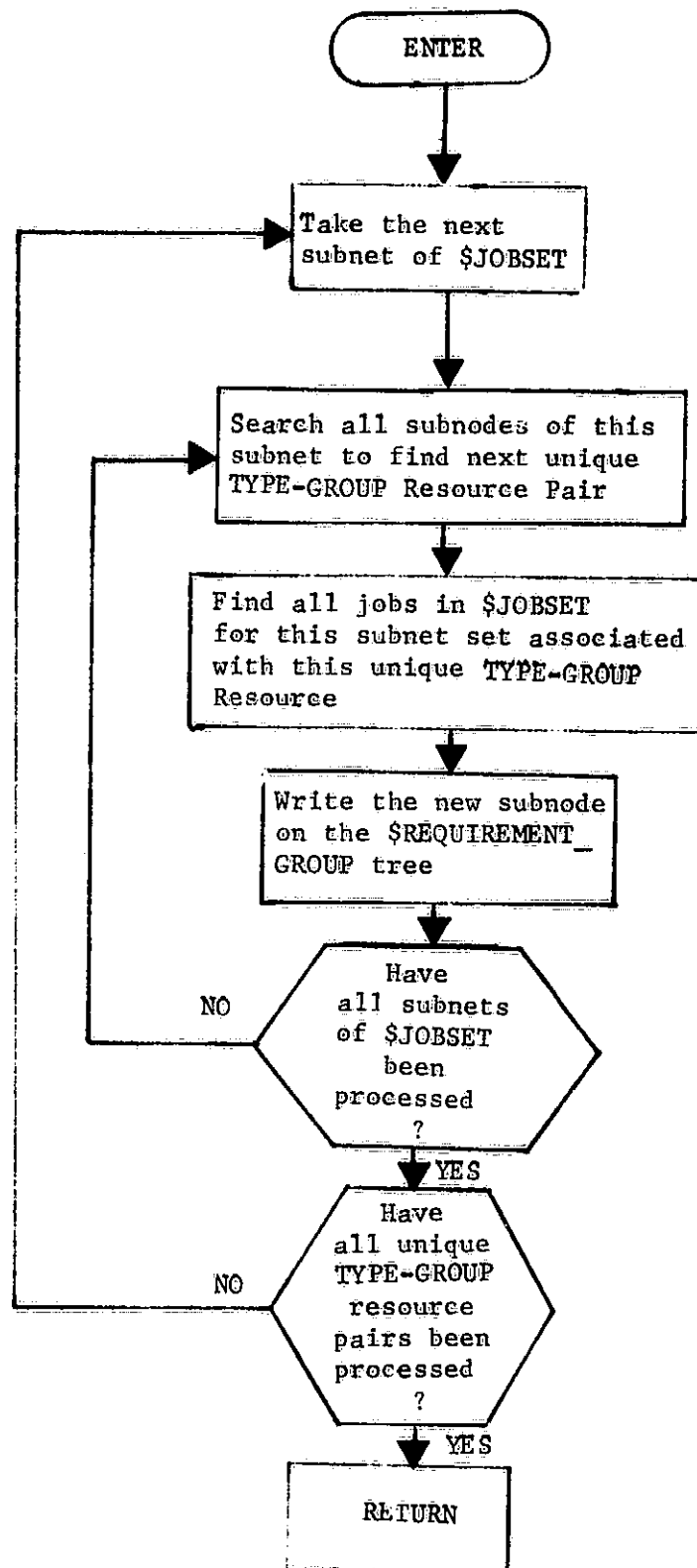


Figure 3-6 Processing of Commonality Groups Logic



case, the algorithm then progresses to the next resource type required by this job.

If this job does belong to a requirement group for this resource type, then the explicit allocation is more complicated. The resource that is to be explicitly allocated must be examined not only to see if it satisfies the requirements demanded by this job, but also to see that it satisfies the requirements demanded by all jobs belonging to that commonality group. The first step is to determine if any specific allocations have already been made by the user. If they have been, then it is necessary to determine if they satisfy the requirements of the other jobs in the requirement group. If they do, then they are assigned to the jobs and all bookkeeping is done. If they do not and contingency resources also fail, then that fact is output and the algorithm proceeds to the next resource type for the original job being handled.

If the user has not made previous explicit allocations, then all resources are examined to see if any are available which satisfy the total needs of all jobs belonging to the requirement group. If none exist, even after contingency levels are examined, then that fact is output and the algorithm proceeds to the next resource type for the original job. If there is an explicit resource available which satisfies the requirements of all the jobs in the requirement group, it is assigned to the jobs and all the necessary bookkeeping is done and the algorithm proceeds as above. Once all the resource requirements have been examined for a given job, then the algorithm proceeds to the next job in the schedule.

#### 3.4.4 Computational Experience

3.4.4.1 Test Case Description - To illustrate the use of the Timeliner and Allocator, consider the following example problem. A payload going to low earth orbit is to be launched between time = 13 and time = 35 with time measured in days and current time = 0. To accomplish the mission, the following activities must occur. An orbiter must be outfitted to handle this particular payload and the payload installed in the orbiter. Call the activity "Prepare Orbiter." The SRB's must be attached to the tank (Prepare Booster). The booster (tank plus SRB's) and the orbiter must be mated (Mate). The entire Shuttle must be brought to the launch pad and launched (Launch). The Shuttle must take the payload to orbit, deliver it and return (Mission).

The launch preparations and flight can be presented as a network (see Figure 3-8), with the launch window constraint being handled using direct constraints on the early and late start time of the activity Launch. Initially, the only resources which will be considered are the orbiter, the tank, the two SRB's and the launch pad. The orbiter is required by all activities except Prepare Booster, the tank and SRB's by all activities except Prepare Orbiter, the launch pad only by the activity Launch.

Ordinarily, the scheduling for this flight would take place as part of the scheduling for many flights. Therefore, it must be insured that the orbiter, SRB's and launch pad are all in a useable condition before they are employed. This can be accomplished by including the refurbishment activities as part of the network for a single flight.

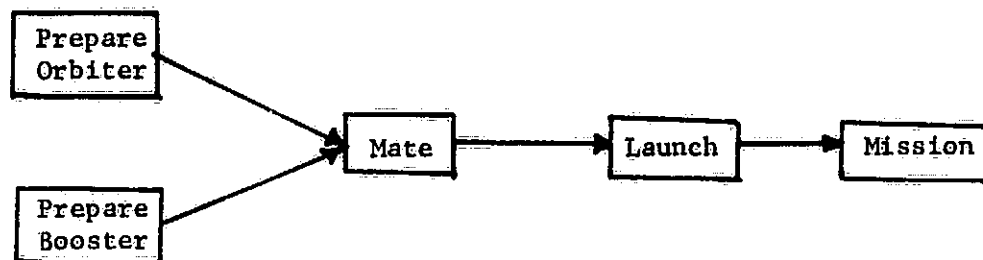


Figure 3-8. Sample Flight Network

The convention can be established that the equipment is either refurbished or unrefurbished when it is received by the particular flight and must be returned in the same condition. Here it is assumed that the equipment is refurbished so that the refurbishment activities can be added at the end of the network (see Figure 3-9).

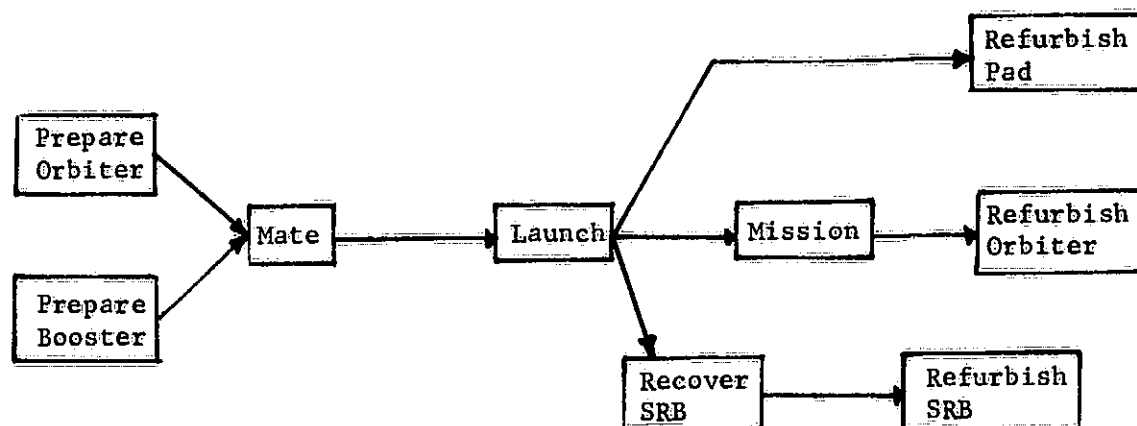


Figure 3-9. Sample Flight Network With Refurbishment Activities

Two difficulties arise when trying to decide the project scheduling resource requirements for the network of Figure 3-9. Once the activity Prepare Orbiter has begun, the orbiter is not really available to other flights until it has been refurbished. If the problem is modeled so that Prepare Orbiter requires one orbiter initially and finally, Mate requires one orbiter, etc., the orbiter would be returned to the pool between the occurrences of these activities, and other flights might be scheduled based on its presence. Also, the activities within a single flight would be scheduled as soon as any orbiter was available, not necessarily the same one. To overcome these problems, the following model can be used. The first activity to require the orbiter (in this case Prepare Orbiter) has a resource requirement whose initial quantity is one and whose final quantity (the amount returned to the pool) is zero. Thus, once the preparations have begun, the orbiter is unavailable to any other flights and if it is not refurbished, it never will be. The activities Mate, Launch and Mission no longer require an orbiter. If an orbiter was available for the scheduling of Prepare Orbiter and their predecessors have been scheduled, they can be assured that the conditions necessary for their occurrence have been met. The activity Refurbish Orbiter has a resource requirement whose initial value is zero and final value is one. It may occur as soon after the mission is complete as all its resource requirements can be met. When it is finished the orbiter will again be available to other flights. In this way flights will be timelined only when the same orbiter is available for all the

activities of the flight which require it and the requirement to assign the same orbiter to the timed activities can be met. A similar task can be taken for the other refurbishable resources.

It should be noted that if a time-transcendent heuristic time-liner (one which does not step sequentially through time) is used with this modeling, it is easy to encounter conditions under which activities cannot be timed. Since time-transcendent timeliners schedule activities as they occur in a prioritized list, high priority activities which occur late on the timeline will be placed on the timeline first using their required resources. Lower priority activities might be able to use and refurbish the same resources before the later activities need them. However, the lower priority activity which first requires the resources cannot be scheduled unless the resource is available for the remainder of the scheduling horizon, which it is not. A parallel heuristic like Project Scheduling Timeliner steps through time scheduling the eligible activities and thus avoids the problem.

The modeling used for the refurbishable resources in the sample flight network is sufficient to handle the problem of using the same resource for a set of activities when no activity outside the set may use the resource until the set is complete. A different problem arises when the same resource is needed but it is free for use by outside activities between the occurrences of the activities in the set. For example, it may be required that the same technician who performs one group of tests perform a later group of tests. The technician is free to work on other activities between the tests. No satisfactory modeling

has been found to insure that the same technician is used for the tests and to allow the technician to be part of a pool between the tests.

Besides the refurbishment activities, the activity Recover SRB was added to the original sample flight network (see Figure 3-9). This activity presents the additional difficulty that it must be scheduled almost immediately after the launch so that the SRB's are not damaged. The constraint cannot be expressed directly within the Timeliner model. However, it could be handled as follows. Suppose that to recover the SRB's requires two ships. The ships take three days to sail to their positions and three days to pick up the SRB's and return. The constraint could be thought of as a requirement not to schedule the launch until there are two ships available for the three days preceding and succeeding the launch. A resource requirement for ships could be added to the activity Launch. Assuming that the actual liftoff occurs at the end of Launch, the interval over which the ships are required would be -2 to +4 measured from the start of Launch. The activity Recover SRB is added to the network requiring no resources so that it will be scheduled as soon as its predecessor Launch is completed.

The modeling presented here is sufficient to provide good schedules for the sample flight preparations scheduling problem with almost no fine tuning. However, it is not implied that all constraints can be manipulated into a project scheduling format, hence the need for iteration and human intervention as illustrated in Figure 3-10.



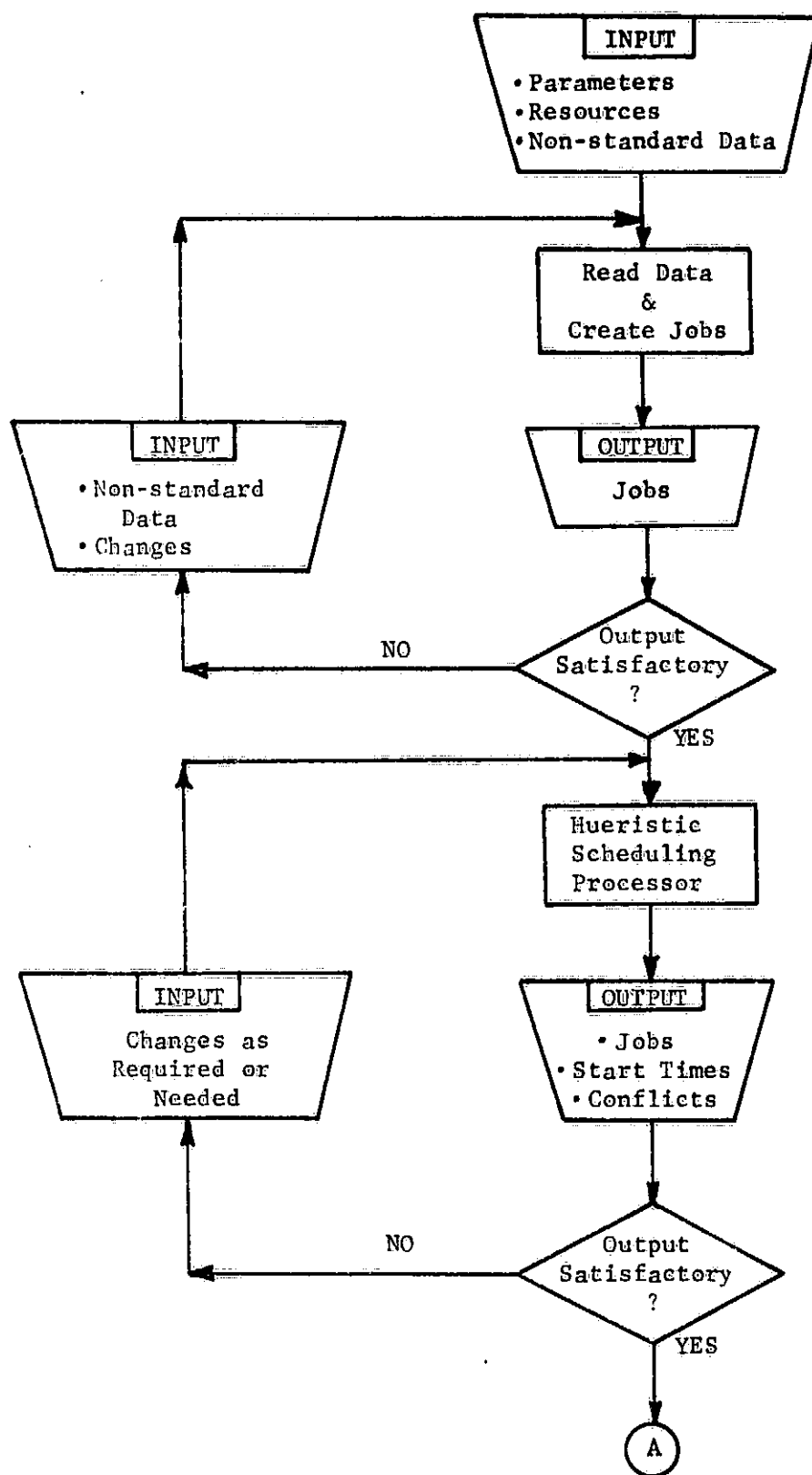


Figure 3-10 Man/System Interface for Demonstration of Scheduling System

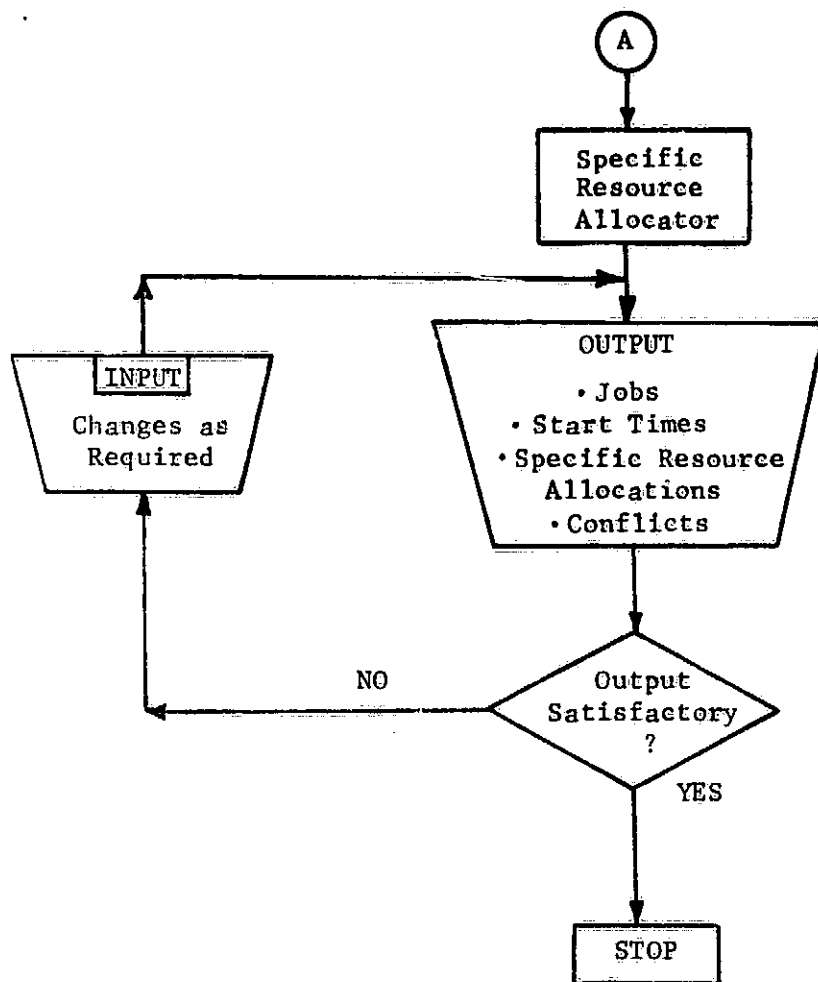


Figure 3-10 (Conclusion)

To illustrate their use, the Timeliner and Allocator were executed manually on the following problem. With the initial time being zero, three flights must be scheduled within the launch windows shown in Table 3-1.

Flight	Launch Windows	
	start time	end time
Flight 1	14	36
Flight 2	16	24
Flight 3	20	25

Table 3-1

#### Example Problem Launch Windows

Each flight requires scheduling of all the activities of Figure 3-9. The duration of each activity and the resources which must be present to accomplish it are shown in Table 3-2. The activity durations and the launch window times are given in days.

Activity	Duration	Required Resources					
		Orbiter	Tank	SRB	IC	Pad	Ship
Prepare Orbiter	4	1					
Prepare Booster	9		1	2	1		
Mate	4	1	1	2	1		
Launch	1	1	1	2		1	
Mission	1	1					
Refurbish Pad	3					1	
Refurbish Orbiter	7	1					
Recover SRB	3			2			2
Refurbish SRB	80			2			

\*IC - Integration Cell, gantry-like device used to support the space shuttle parts during assembly.

Table 3-2. Durations and Resources Needed for Example Problem Activities

The number of each resource available for use in scheduling the activities is shown in Table 3-3. All resources are assumed to be refurbished and ready for use at time zero and available to the three flights throughout the scheduling horizon of the problem.

Resource	Number Available
Orbiter	2
Tank	3
SRB	6
IC	2
Pad	1
Ship	2

Table 3-3. Resources Available for Example Problem

3.4.4.2 Test Results - The Timeliner was first executed with the resources modeled exactly as they appear in Table 3-2. In addition to the ordinary precedence constraints the launch window start and end times were used to provide temporal constraints for the activity Launch. Since the actual liftoff was assumed to occur at the end of Launch, the early start and late start constraints which were put on the activity Launch for each flight were one day earlier than the launch window constraints. The duration of the activity Refurbish SRB was much longer than those of other activities and enough SRB's were specified in the problem statement so that no SRB was needed for more than one flight. Therefore, the activity Refurbish SRB was eliminated. The

activity Dispatch Ships (duration = 3) requiring two ships was added as a predecessor of Recover SRB to allow time for the ships to arrive on station before the SRB's are dropped. The timeline generated using this modeling is shown in Figure 3-11. While the explicit precedence, early start and late start constraints were met, the timeline is far from satisfactory. All three flights must share the same pair of recovery ships. However, the ships are dispatched for flights 2 and 3 before they have returned from recovering the SRB's for flight 1. The activity Prepare Orbiter for flight 3 is scheduled after both orbiters have been prepared for flights 1 and 2 but before either orbiter has been launched and refurbished. For the timeline, no specific allocation of resources can meet the requirement that the same orbiter and the same integration cell must be used throughout each individual flight. Indeed, the mission for flight 2 is delayed 4 days after its launch so that the orbiter for flight 2 can be used in the Mate activity for flight 3.

In order to produce a timeline for which satisfactory specific resource allocations could be made, the resource modeling discussed in the previous section was employed. Then, since the requirement to allow time for the ships to sail to the SRB recovery point is met by making the ships a resource required by Launch, the activity Dispatch Ships was eliminated. The resource modeling used for this execution of the Timeliner is shown in Table 3-4. The timeline generated is shown in Figure 3-12.

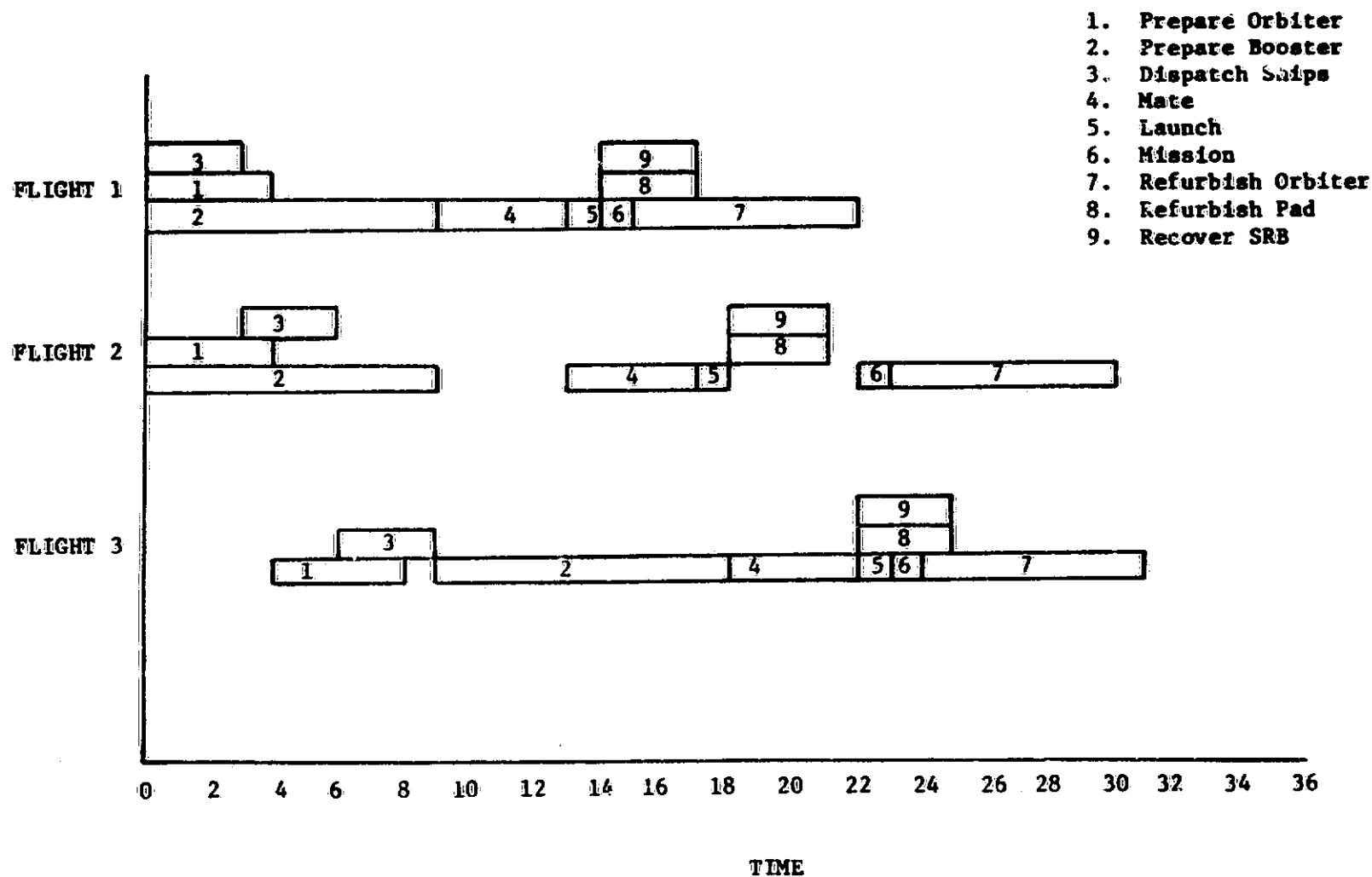


Figure 3-11 Timeline Generated Using Resource Modeling of Table 5-2

1. Prepare Orbiter
2. Prepare Booster
3. Mate
4. Launch
5. Mission
6. Refurbish Pad
7. Refurbish Orbiter
8. Recover SRB

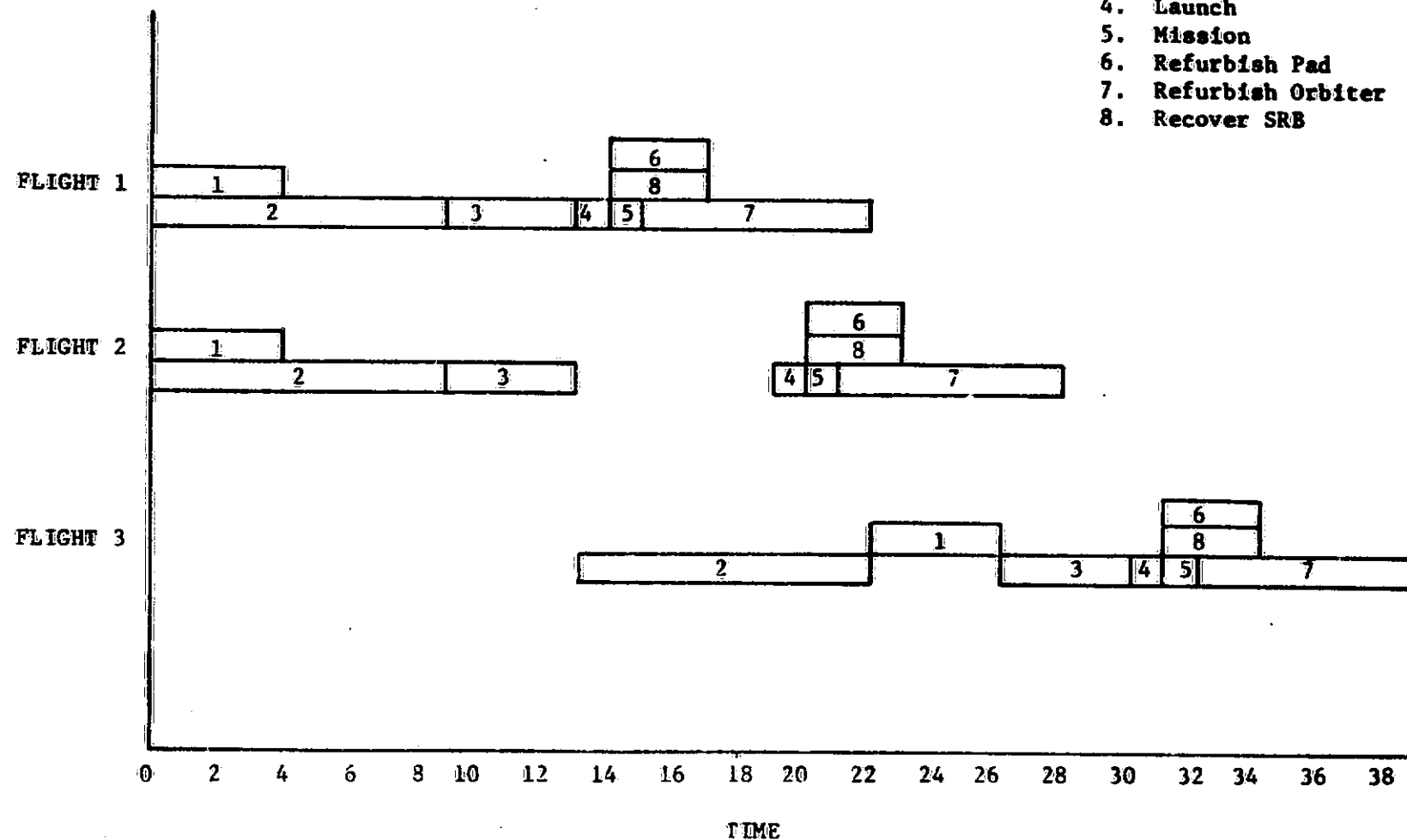


Figure 3-12 Timeline Generated Using Resource Modeling of Table 3-4

Activity	Duration	Required Resources (initial/final)					
		Orbiter	Tank	SRB	IC	Pad	Ships
Prepare Orbiter	4	1/0					
Prepare Booster	9		1/0	2/0	1/0		
Mate	4				0/1		
Launch	1					1/0	2*
Mission	1						
Refurbish Pad						0/1	
Refurbish Orbiter	7	0/1					
Recover SRB	3						

\* Interval over which 2 ships are required is -2 to +4 relative to start of Launch

Table 3-4  
Modeling Used for Second Timeliner Execution

The timeline of Figure 3-12 has none of the unacceptable traits of the first timeline. The orbiter used for flight 3 is not prepared until the orbiter used for flight 1 has flown its mission and been refurbished. The launches are separated by enough time so that the recovery ships can leave port, sail to the SRB recovery area, recover the SRB's and return them to port before the ships are needed for the next launch. It would even be possible to assign the same orbiter and integration cell for all the activities that need them within each flight. However, Launch for flight 3 is scheduled for time = 30 making its liftoff at time = 31. The launch window for flight 3 ends at time = 25.

The second execution of the Timeliner has provided a timeline which meets most of the constraints and focuses attention on flight 3, which has the smallest launch window. Since flight 1 has the latest ending launch window, it would be logical to delay flight 1 until



flight 3 has had a chance to use the critical resources. This could be accomplished by fixing the start times of Prepare Orbiter and Prepare Booster for flight 3 at time = 0 or by making those activities predecessors of Prepare Orbiter and Prepare Booster for flight 1. The third execution of the Timeliner was made using the same modeling as the second execution with Prepare Orbiter for flight 3 added as a predecessor to Prepare Orbiter for flight 1 and Prepare Booster for flight 3 added as a predecessor of Prepare Booster for flight 1. The timeline generated using this modeling is shown in Figure 3-13. This timeline avoids the unsatisfactory aspects of the first timeline and meets both the precedence and launch window constraints. Therefore, the Allocator was used in an attempt to assign specific resources to the activities as timeline/ in Figure 3-13.

The results of the allocation are shown in Table 3-5. For the modeling used to generate the Figure 3-13 timeline, it is sufficient to specify that the activity which makes a resource unavailable and the activity which returns that resource to availability use the same resource item. The allocation of Table 3-5 meets this requirement. Thus, all the temporal constraints and resource requirements of the sample problem have been met.

The philosophy used in solving the sample problem was to account for as many constraints as possible through modeling (second execution), isolate the remaining violations (flight 3 launch window) and manipulate the inputs to the Timeliner to allow the constraints to be satisfied. While the specific allocation made after the third Timeliner execution

1. Prepare Orbiter
2. Prepare Booster
3. Mate
4. Launch
5. Mission
6. Refurbish Pad
7. Refurbish Orbiter
8. Recover SRB

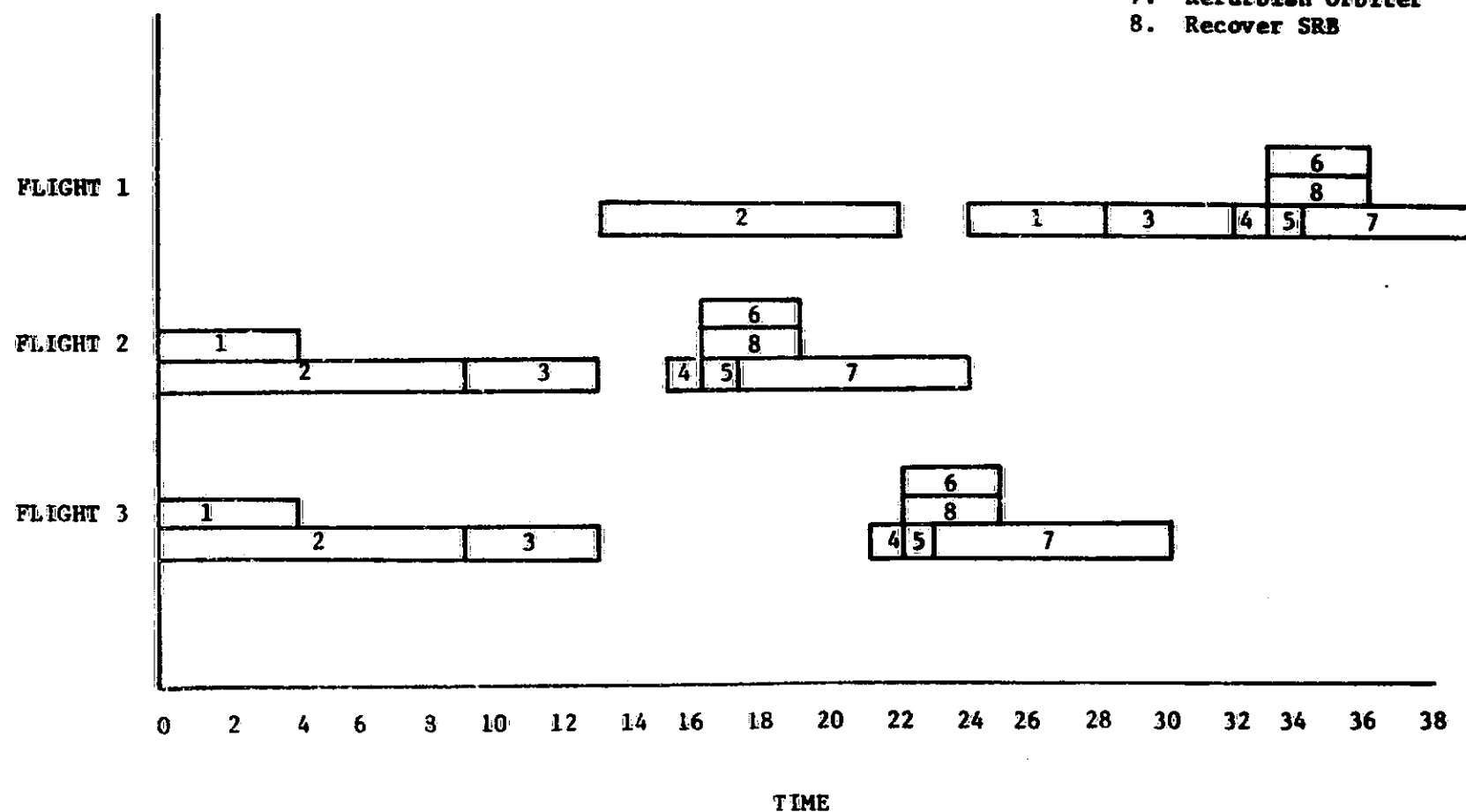


Figure 3-13 Timeline Generated Forcing Flight 3 to Occur Before Flight 1

ACTIVITIES	RESOURCES					
	ORBITER	TANK	SRB	IC	PAD	SHIP
<b>Flight 1</b>						
Prepare Orbiter	#1					
Prepare Booster		#3	#5, #6	#1		
Mate				#1		
Launch					#1	
Mission						
Refurbish Pad					#1	
Refurbish Orbiter	#1					
Recover SRB						#1, #2
<b>Flight 2</b>						
Prepare Orbiter	#1					
Prepare Booster		#1	#1, #2	#1		
Mate				#1		
Launch					#1	
Mission						
Refurbish Pad					#1	
Refurbish Orbiter	#1					
Recover SRB						#1, #2
<b>Flight 3</b>						
Prepare Orbiter	#2					
Prepare Booster		#2	#3, #4	#2		
Mate				#2		
Launch					#1	
Mission						
Refurbish Pad					#1	
Refurbish Orbiter	#2					
Recover SRB						#1, #2

TABLE 3-5. Specific Resource Allocation  
for Timeline of Figure 3-13

was satisfactory immediately, other problems may require further  
Timeliner or Allocator iterations based on information gained from  
unsuccessful specific allocations.

### 3.5 A SYSTEM CONFIGURATION COMPATIBLE WITH THE MAN-COMPUTER STRATEGY

#### 3.5.1 Dialog Methods for Scheduling

Selection of a dialog method is an important element in the functional design of an interactive computer system, since it may affect the choice of hardware, grossly impacts the system software design, and has some potential impact on the underlying computational software. Many dialog methods are available, differing in the type of information which can be exchanged between man and computer, the speed of the conversation, and the knowledge and skills required of the user. Selection of a dialog method may involve a trade-off between desirable functional capabilities of the dialog and cost impact on the computer system.

Certain dialog properties are clearly desirable in an interactive scheduling system. First, it is highly desirable that the dialog provide for graphical displays, since much of the information needed by the scheduler in order to make his decisions is naturally expressible in a graphical form which is already comprehensible to him. It is absolutely necessary that the dialog be truly interactive (i.e., that the communication can be conducted at a rapid, conversational pace), since much of the user's task consists of exploration, via a series of displays, of a fairly large information space. When the user's task includes a great deal of searching, via many displays, for a variety of information, it is necessary that the system provide a rapid means for construction of display queries. The system will probably have to support infrequent users -- people who are concerned

with scheduling problems, but not on a daily basis. In order to provide for both infrequent users and very proficient users, the system should either have a computer-initiated dialog (which guides the user through his options) which is rapid and unobtrusive enough that it does not interfere with the experienced user, or it should provide separate dialog methods for experienced and inexperienced users. It is desirable that the system support parallel displays which allow the user to visually correlate information in related displays. A mechanism should be provided which allows the user to easily specify information already being displayed as part of the query for the next display. Names of particular jobs, resources, etc., will often be used in this way. Finally, regardless of the specific dialog method selected, it is highly desirable that the system be fully self-tutorial, providing interactive instruction for the new or infrequent user.

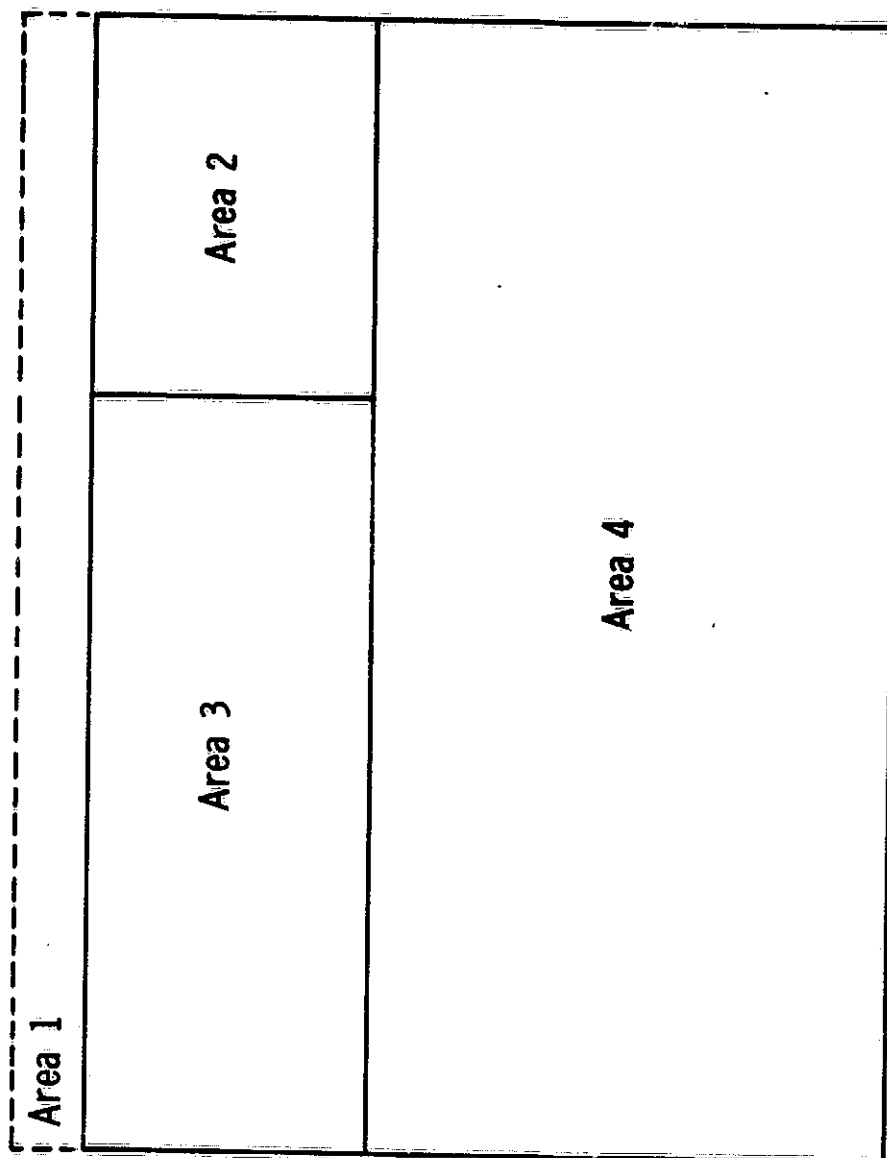
A dialog method which satisfies these requirements, requires an interactive graphics terminal equipped with a light-pen and alphanumeric keyboard. The use of a vector graphic device (one in which lines are actually drawn on the CRT from one point to another), rather than some other method of line generation is not a strong requirement of this application, since most of the lines are horizontal or vertical and would be reasonably well represented if generated by incremental plotting or dots. It is necessary, however, that the device employ a refreshed display in order to allow the necessary flexibility and rapid exchange of option lists (light buttons) and other display

elements. It is also necessary that the display be associated with a local minicomputer, since available communications facilities and operating systems for the host computer cannot provide the necessary speed for truly interactive command construction, and such use would unnecessarily burden the host computer anyway.

The use of a graphical device equipped with a light-pen allows a very natural solution to two of the problems posed by the requirements. First, it provides a medium in which computer-initiated dialogs can be conducted without interfacing with the rapid use of the system by an experienced user. Second, it provides a ready mechanism by which the user can select information already on the screen from some previously requested display as the value of a parameter which must be specified as part of the command he is currently constructing. Thus, if he desires information about a particular job, and the name of that job is currently being displayed as part of a timeline display, for example, he can simply select the displayed name with the light-pen to indicate which job he is interested in.

A basic man-computer interface design framework will be described within which an interactive scheduling system could be implemented. The design provides for the use of both light-pen and keyboard inputs, usually at the user's option and it provides some standard techniques for use of the graphical display and for conduct of the man-computer dialog.

The display is divided into four areas, as shown in Figure 3-14. Area 1 is reserved for the command which the user is constructing, or



**Fig. 3-14 Segmented Graphical Display**



--	--	--	--	--	--	--	--

has just executed. Area 2 contains instructions to the user and lists of options from which he can select. These lists take the form of "menus", or "light-button lists", which allow selection with a light-pen. Areas 3 and 4 are available for system displays, both graphical and textual. Area 4, the largest area, is the primary display area, and will contain such information as timeline displays, tutorial information, etc. Displays which are needed at the same time as the major displays (e.g., resource profile information which should be viewed simultaneously with the corresponding timeline information) will be displayed in area 3.

Figure 3-15 is a photograph showing how an actual display on the system might appear. A simple timeline display is shown in area 4, with a resource profile display in area 3. A sample light-button list is shown in area 2. Let us suppose that the user, viewing such a display, wants to display information about a particular job, for example job JOB\_2. He would first select the word DISPLAY with the light-pen. A new light-button list would then appear in area 2, showing him the options available to him for information displays. At the same time, the word DISPLAY would appear in area 1. During the command construction process, area 1 will continually display that portion of the command which has already been constructed.

The light-button list which appears in area 2 after DISPLAY is selected might contain such options as JOB, RESOURCE, TIMELINE, etc. The user can now select the desired option from this list, in this case JOB. Having done so, he is then instructed, via information in

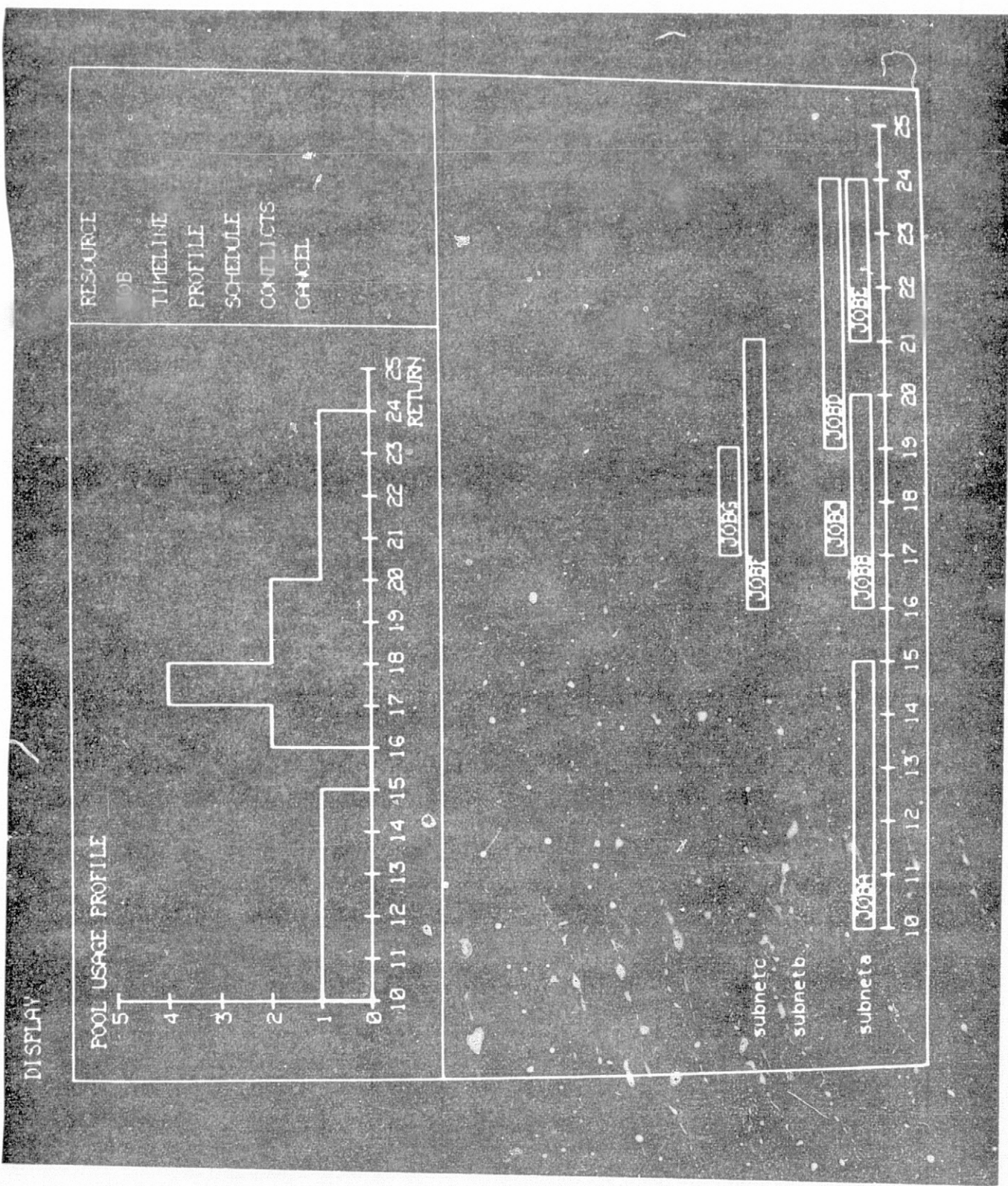


Figure 3-15 A Sample Display

ORIGINAL PAGE IS  
OF POOR QUALITY

area 2, to indicate which job he is interested in. He can indicate this information (and, in fact, could have selected DISPLAY and JOB) by typing the desired input from the keyboard. There is an easier way, however, and one which avoids the necessity for switching from the light-pen to the keyboard and back. Since the job name JOB\_2 is already displayed on the CRT, he can simply select it with the light-pen.

At each point in the command construction process, the user has the option to CANCEL, that is, to delete the partially completed command and start over. In addition, any time the command is in a state which is legal as a complete command, he has the option to select EXECUTE. With the exception of a few functions which can be executed entirely by the minicomputer, selection of EXECUTE is the mechanism for causing the command to be transmitted to the host computer. Up to this point, however, the entire man-computer dialog has been conducted under control of the minicomputer.

The HELP light-button on the main light-button list is a mechanism whereby the user can request the system's TUTOR, a document which can be displayed on-line in order to train the beginning user in the use of the system, and to provide an easy review mechanism for the experienced user who has forgotten how to perform some particular function. The display of the TUTOR should be entirely under the control of the minicomputer in order to allow beginning users to become familiar with the system without incurring host computer connect costs. The TUTOR should provide a table of contents with a direct access mechanism,

preferably using the light-pen, to allow the user to go directly to that part of the document which contains information relevant to his immediate needs. It should also provide a simple mechanism for returning from the TUTOR to the user's previous display.

### 3.5.2 Host/Mini-Computer Configuration for Interactive Scheduling

The kind of dialog design described in the preceding section requires that most of the dialog control logic reside at the interactive graphic terminal. The resulting division of responsibility between the host computer and the minicomputer has led to the development of a candidate system design which will be discussed in this section.

A minicomputer software design which could support the dialog methods discussed in the previous section is described below. The hardware components are:

- CPU and core memory
- Random access I/O device
- CRT with light-pen
- Keyboard
- Communication line interface

and are depicted in Figure 3-16. A possible core configuration is shown in Figure 3-17.

The resident area contains the code necessary for driving all of the peripheral equipment as well as utility routines to support the various display generators. The overlay area is used either for the code to generate a graphical display from a tree data structure, or for the code to construct a command for processing by the host computer.

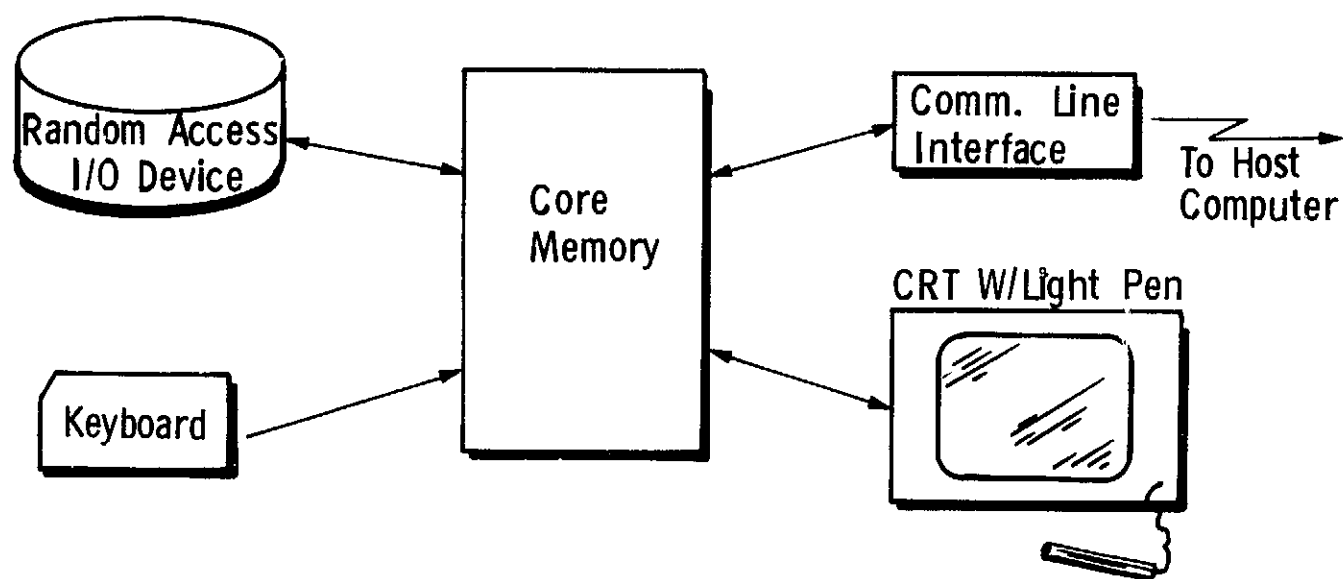


Fig. 3-16 Minicomputer-Related Hardware Components

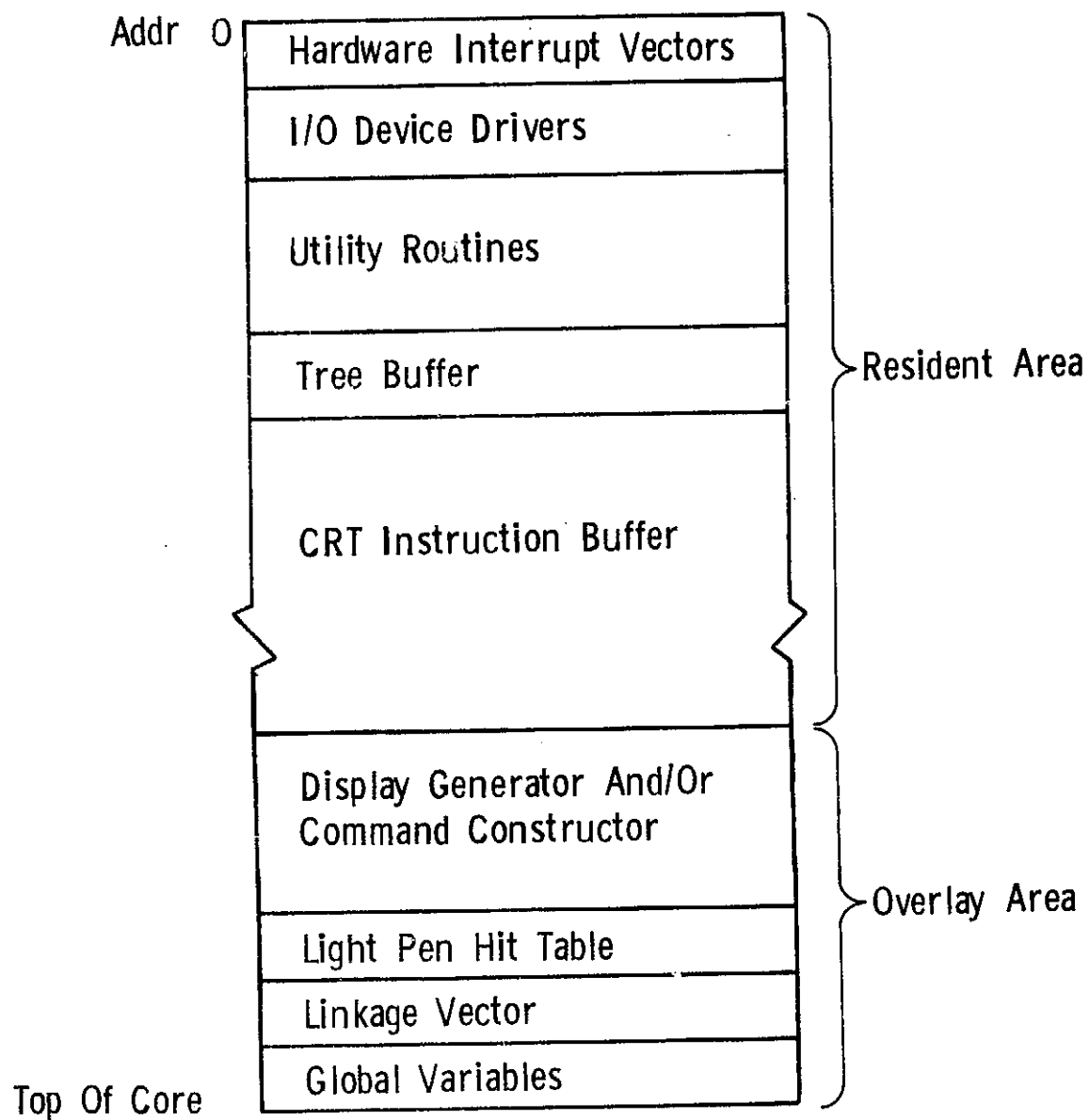


Fig. 3-17 Minicomputer Core Layout

Tasks supported by the resident software include:

- 1) Memory management for the CRT instruction buffer. A display generator overlay will typically ask that the current display in area 3 or 4 be deleted, then ask for assignment of an area in the CRT instruction buffer where it can build a new sequence of CRT instructions, and finally ask that the new instructions be executed to bring up the display on the CRT.
- 2) Command line display and transmission. The command line is displayed in area 1 and may actually require more than one line on the screen since commands may contain as many as 150 characters.

Operations which must be supported on the command line are:

- addition of each character typed at the keyboard to the command line.
  - a rubout key for deleting typed characters.
  - addition of character strings to the command line when a light button is selected.
  - transmission of the command line to the host computer.
  - clearing the command line.
- 3) Receiving data from the host computer. A typical response to a command sent from the mini to the host computer is for a reply to be sent back to the mini in the form of a compressed tree data structure. An identifier on the front of the data structure can be used to determine which overlay is to be loaded to process the reply. When the tree transmission is complete, the appropriate overlay is loaded and executed. Trees sent from the host computer

may have to overflow to mass storage since a large tree may be 3000 characters or more. In addition, four trees may be active at the mini at any given time, two for area 3 and two for area 4. Provision must be made for storing all four trees on mass storage and loading the first segment of any one on request. Any routines which scan trees must have provision for detecting the end of the buffer and requesting loading of the next tree segment from mass storage.

- 4) Light pen hit processing. One method of handling light pen hits is to associate a unique identifier with each light button which may appear on the screen. This identifier can then be used by the resident software as an index to the light pen hit table loaded with the current overlay. The light pen hit table can then point to the routine for processing the hit. In cases where the current overlay does not contain the necessary code for processing a light button displayed by another overlay, the light pen hit table can contain the identifier of the overlay which must be loaded.
- 5) Mass storage management. Data which must be stored on the random access I/O device include:
  - textual information for the tutor. This is estimated to be about 250,000 characters divided into "pages" which can be displayed in area 4.
  - display generation overlays.
  - four data trees of about 2000 characters each must be stored on the mass storage device.



- two files should be maintained containing the previous displays for areas 3 and 4. This will allow the user to swap back and forth quickly between the previous display and current display in these areas.

Because the minicomputer controls the command construction process and generates the graphical displays from information stored in tree structures, the burden on the host computer is minimal. Very little modification is required to convert PLANS batch scheduling programs into a form compatible with interactive execution under IBM's Time-Sharing Option, and compatible with the minicomputer software system. The host scheduling software must be capable of reading and writing trees in compressed form (a simple modification to the PLANS language itself). The only significant additional software required is an interpretive execution capability for receiving and parsing scheduling commands and causing the appropriate PLANS programs to execute. This capability can be achieved most easily through the use of the same Translator-Writing System which has been used to implement PLANS. The interactive scheduling command language is described via an augmented grammar which defines that language in terms of PLANS subroutine calls. The grammar is then input to the Translator-Writing System, which generates an interpreter for the interactive scheduling system. This interpreter, together with an appropriate lexical analyzer, the PLANS scheduling subroutines, and appropriate PLANS support software constitute the host computer software necessary for interactive scheduling.